Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

Understanding Sorting Techniques Using C++

Zobair Ullah

Sam Higginbottom Institute of Agriculture, Technology & Sciences, Allahabad, India

Abstract: **The paper is intended to introduce some well known sorting techniques using array and C++. The paper briefly defines and describes the different sorting techniques available and well known to us giving suitable example.**

Keywords: **Sorting, Quick sort, Bubble sort, Insertion sort, Selection sort, Merge sort, Heap sort, Radix sort, Shell sort.**

1. INTRODUCTION

The term sorting basically means the process of putting things in order. This technique is considered as an important tool of making things in proper order and in a way a person prefers to organise things. These days, this technique is increasingly employed in almost all types of processes, methods and procedures. Sorting technique is generally used by business firms like manufacturing industries, schools, markets and libraries. Sorting technique is also used to assess the navigation hierarchy of a website. At present, the most used orders are numerical order and lexicographical order. It is also useful for canonicalizing data and producing human readable output. Therefore, there is a need to study and learn some well known sorting techniques used by computer programmers to solve some real world problems easily and effectively. So, let us discuss, define and understand some well known sorting technique.

2. DEFINITION

The term sorting defined by different agencies/sources/programmers of the world is as under:

Sorting-------- refers to ordering data in an increasing or decreasing fashion according to some linear relationship among the data items.

OR

Sorting is a process of arranging data items (numbers/strings) in some sequence and/or in different sets or groups.

OR

Sorting is the process of putting a list or a group of data items in a specific order.

3. TYPES OF SORTING

There are basically two categories/types of sorting:

3.1 Internal sorting ------- refers to a kind of sorting in which the number of objects is small enough to fit into the main memory.

3.2 External sorting -------- refers to a kind of sorting in which the number of objects is so large that some of them reside on external storage during the sort.

Some well known sorting techniques are as under:

1. Quick sort 2. Bubble sort 3. Insertion sort 4. Selection sort 5. Merge sort 6. Heap sort7. Radix sort 8. Shell sort

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

Now, let us understand and explain all the above sorting techniques in detail.

4. QUICK SORT

4.1 Quick sort: refers to a sorting technique that sequences a list by continuously dividing the list into two parts and moving the lower items to one side and the higher items to the other. It is also called partition –exchange sort.

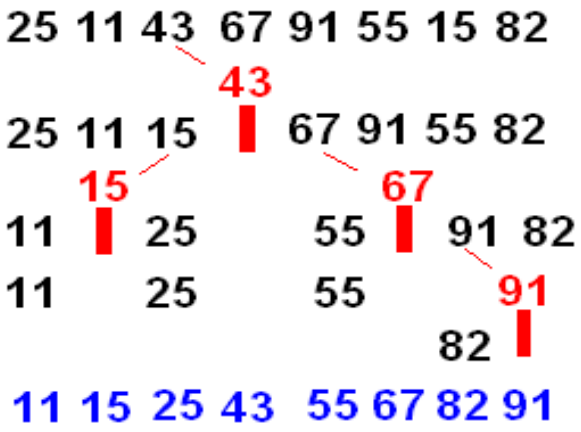

Figure 1: Shows the working of quick sort

This quick sort uses randomly chosen pivots (in red) to keep dividing the list into two until there is only one item on each side of the pivot left.

4.2 The process of quick sort algorithm is:

- Select an element, pivot, from the list.
- Rearrange the elements in the list, so that all elements those are less than the pivot are arranged before the pivot and all elements those are greater than the pivot are arranged after the pivot. Now the pivot is in its position.
- Sort the both sub lists – sub list of the elements which are less than the pivot and the list of elements which are more than the pivot recursively.

4.3 Salient features:

- Quick sort algorithm was invented by C.A.R .Hoare and introduced in the year 1962.
- It is based on the principle of divide and conquer.
- It is easy to implement and considered as a good ”general purpose” sorting technique.
- It uses only a small auxiliary stack.
- It requires only n log (n) time to sort n items.
- It has an extremely short inner loop.
- It is faster than other algorithms.
- The quick sort algorithm is totally based on mathematical analysis.
- It has two phases:- 1) The partition phase 2) The sort phase.
- It is an example of randomization and average case analysis.

The following program illustrates the implementation of the quick sort.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

4.4 A program to sort array elements in ascending order using quick sort method:

```
#include<iostream.h>              //Function prototype
#include<conio.h>
#include<stdlib.h>
int ar[5]={10,5,15,25,20};          // Array declaration statement
int qsort(const void *p, const void *q)
{
return(*(int *)p -*(int *)q);
}
void main()                    //Declaration of main function
{
int a;
clrscr();
cout<<"\n"<<"Array members are:"<<endl;
for(a=0;a<5;a++)
{
cout<<"   "<<ar[a];
}
qsort(ar,5, sizeof( int ),sort);
cout<<"\n"<<"Sorted array members are:"<<endl;   //Accessing sorted array member
for(a=0;a<5;a++)
{
cout<<"   "<<ar[a];
}
getch();                    //freeze the screen until any key is pressed
}
```

Output:

Array members are:

10 5 15 25 20

Sorted array members are:

5 10 15 20 25

4.5 Big O notation:

In computer science, big O notation is used to classify algorithms by how they respond (processing time needed to execute the program or work space required by the program) to changes in input size.

4.5.1 Sorting efficiency:

Sorting techniques mainly depends on two factors: a) the execution time of the program and b) the space taken by the program.

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

4.5.2 The complexity analysis of quick sort is:

Worst Case Time Complexity : $O(n^2)$

Best Case Time Complexity : O(n log n)(simple partition) ***OR*** O(n)(three way partition and equal keys)

Average Time Complexity : O(n log n)

Space Complexity : O(n) auxiliary(naive) ***OR*** O(log n) auxiliary

5. BUBBLE SORT

5.1 Bubble sort: refers to a sorting technique that is typically used for sequencing small lists. It starts by comparing the first item to the second ,the second to the 3rd and so on until it find one item out of order . It then swaps the two items and starts over.

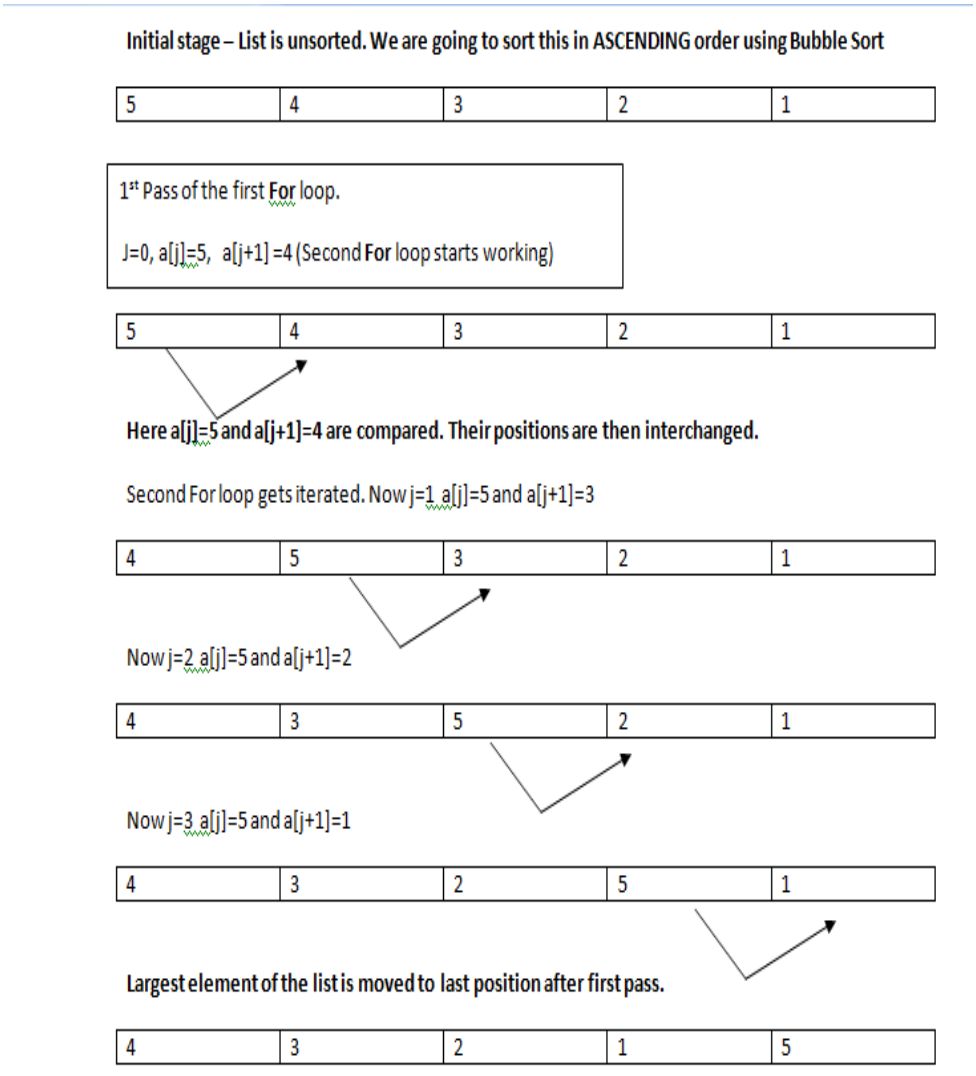

Figure 2: Shows the working of bubble sort

5.2 Salient features:

- Bubble sort is very popular sorting algorithm.
- Bubble sort is an internal exchange sort.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

- Bubble sort works by repeatedly swapping adjacent elements that are out of order.
- The sort may alternate from the top of the list to the bottom and then from the bottom to the top.
- Bubble sort is also known as a sinking sort.
- Bubble sort is intuitive, easy to write and debug.
- Bubble sort consumes little memory.

5.3 The process of bubble sort algorithm is:

- For i =1 to A.length -1
- For j= A.length downto i+1
- If A[j]<A[j-1]
- Exchange A[j] with A[j-1]

OR

5.3.1 For sorting an integer array using bubble sort:

```
for (int i=0; i<n; i++)
{
for (int b=0; b<n-1;b++)
{
if(array[b]>array[b+1])
{
int temp=array[b+1];
array[b+1]=array[b];
array[b]=temp;
}
}
}
```

OR

- In each pass, the first two items in a list are compared and placed in the correct order.
- Items two and three are then compared and reordered, followed by items three and four, and then four and five and so on.
- The sort continues until a pass with no swap occurs.
- As a result, high value items near the beginning of a list move to their correct position rapidly and are called turtles, because they move only one position with each pass.

The following program illustrates the implementation of the bubble sort.

5.4 A program to sort array elements in ascending order using bubble sort method:

```
#include<iostream.h>                    //Function prototype
#include<conio.h>
```

```
void main()
{
int a,b,flag,c;
int ar[5]={3,5,2,1,4};              //Array declaration
cout<<"\n"<<"Array is:"<<endl;
for(a=0;a<5;a++)
{
cout<<ar[a]<<"     ";
}
for(a=0;a<5-1;a++)       //sorting process begins
{
flag=1;
for(b=0;b<(5-1-a);b++)
{
If(ar[b]>ar[b+1])
{
flag=0;
c=ar[b];
ar[b]=ar[b+1];
ar[b+1]=c;
}
}
If(flag)
break;
}
cout<<"\n"<<"sorted list is:"<<endl;     //Accessing sorted array members
for(a=0;a<5;a++)
{
cout<<"    "<<ar[a];
}
getch();
}
```

Output:

Array is:

3 5 2 1 4

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

Sorted list:

1 2 3 4 5

OR

```
#include<iostream.h>            //Function prototype
#include<conio.h>
void main()
{
int a,b,c,n;
n=5;
int ar[5]={10,5,15,25,20};                    //Array declaration statement
cout<<"\n"<<"Array elements are:"<<endl;
for(a=0;a<n;a++)
{
cout<<"  "<<ar[a];
}
for(a=0;a<(n-1);a++)          //sorting begins
{
for(b=0;b<(n-a-1);b++)
{
If(ar[b]>ar[b+1])
{
c=ar[b];
ar[b]=ar[b+1];
ar[b+1]=c;
}
}
}
cout<<"\n"<<"Sorted list is:"<<endl;
for(a=0;a<n;a++)
{
cout<<"  "<<ar[a];
}
getch();                    //freeze the screen until any key is pressed
}
```

Output:

Array elements are:

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

10 5 15 25 20

Sorted list is:

5 10 15 20 25

5.5 The complexity analysis of bubble sort is:

Worst Case Time Complexity : $O(n^2)$

Best Case Time Complexity : $O(n)$

Average Time Complexity : $O(n^2)$

Space Complexity : $O(1)$

6. INSERTION SORT

6.1 Insertion sort ---- refers to a simple sorting algorithm that builds the final sorted array (or list) one item at a time.

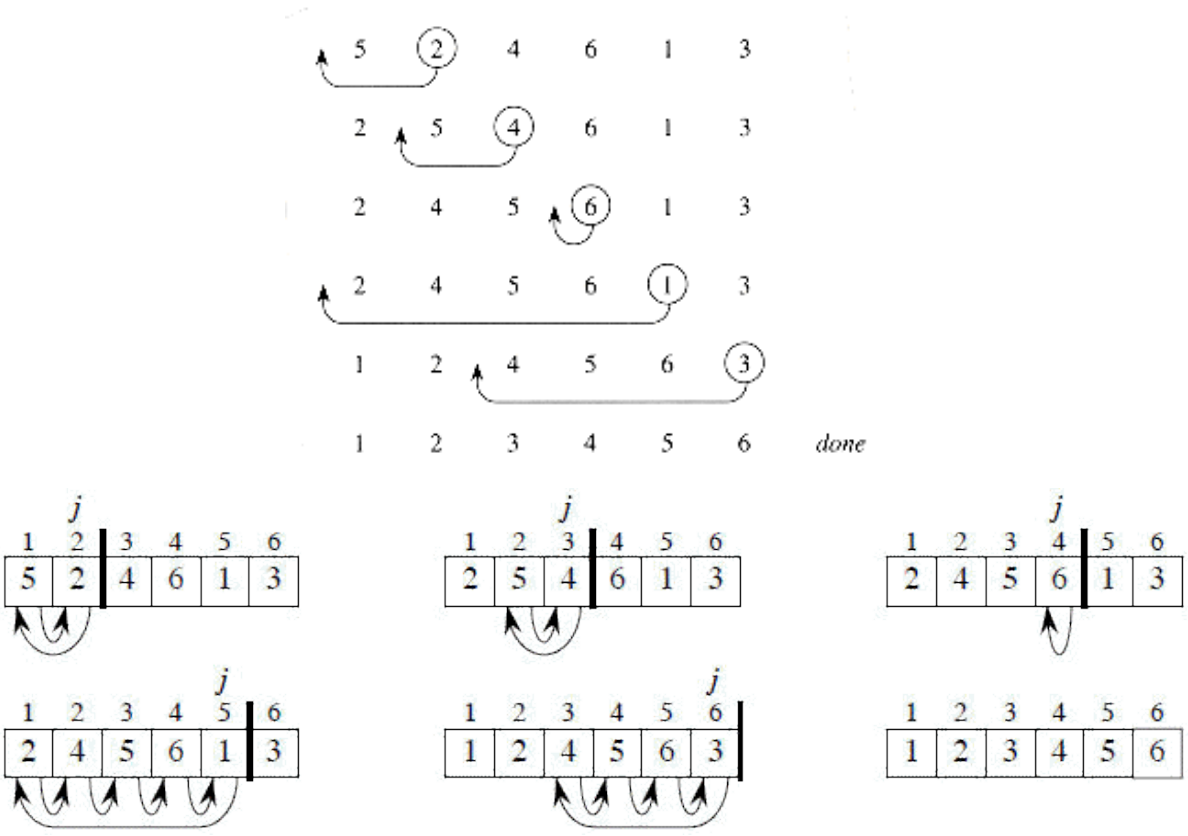

Figure 3: Shows the working of insertion sort

6.2 Salient features:

- Insertion sort consider the elements one at a time, inserting each in its suitable place among those already considered (keeping those sorted).
- Insertion sort builds the sorted sequence one number at a time of an array. The number/element of an array is called key.
- It is less efficient on large list.
- It is simple and easy to implement.

- It is efficient for (quite) small data sets.
- It is stable (i.e it doesn't change the relative order of elements with equal keys).
- Insertion sort is an instance of an incremental algorithm.

6.3 The process of insertion sort algorithm is:

```
For j←2 to length [A]
Do key ← A [j]
{
Put A[j] into the sorted sequence A[1 ……j-1]
}
i←j-1
while i >0 and A[i] >key
do A[i+1]← A[i]
i←i-1
A[i+1]←key
```

OR

```
for ( int i=1; i< ar.length; i++)
{
  int index = ar[i];
int j=i;
while(j>0 &&ar[j-1]>index)
{
ar[j]=ar[j-1];
j--;
}
ar[j]=index;
}
}
```

The following program illustrates the implementation of the insertion sort.

6.4 A program to sort array elements in ascending order using insertion sort method:

```
#include<iostream.h>                    //Function prototype
#include<conio.h>
void main()                    //Declaration of main function
{
int a,b,c,n;
n=5;
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

```
int ar[5]={10,5,15,25,20};          //Array declaration statement
cout<<"\n"<<"Array elements are:"<<endl;
for(a=0;a<n;a++)
{
cout<<"   "<<ar[a];
}
for(a=1;a<=n-1;a++)          //Insertion sorting begins
{
b=a;
while(b>0 && ar[b]<ar[b-1])
{
c=ar[b];
ar[b]=ar[b-1];
ar[b-1]=c;
b--;
}
}
cout<<"\n"<<"Sorted list is:"<<endl;   //Accessing sorted array members
for(a=0;a<n;a++)
{
cout<<"   "<<ar[a];
}
getch();                //freeze the screen until any key is pressed
}
```

Output:

Array elements are:

10 5 15 25 20

Sorted list is:

5 10 15 20 25

7. ARRAY AND SWAPPING

7.1 A program to sort array elements in descending order using swapping method:

```
#include<iostream.h>                    //Function prototype
#include<conio.h>
void main()
{
```

```
int ar[5]={5,10,15,20,25};            // Array declaration statement
int a,b,n,end;
n=5;
end=n-1;
cout<<"\n"<<"The array elements are:"<<endl;
for(a=0;a<n;a++)
{
cout<<"    "<<ar[a];
}
for(a=0;a<n/2;a++)                    //swapping begins
{
b=ar[a];
ar[a]=ar[end];
ar[end]=b;
end--;
}
cout<<"\n"<<"Reversed array is: "<<"\n";    //Accessing sorted array members
for(a=0;a<n;a++)
{
cout<<"    "<<ar[a];
}
getch();                              //freeze the screen until any key is pressed
}
```

Output:

The array elements are:

5 10 15 20 25

Reversed array is:

25 20 15 10 5

7.2 Array and copying elements to second array:

7.2.1 A program to arrange array elements in descending order by copying elements of first array to the second array:

```
#include<iostream.h>                    //Function prototype
#include<conio.h>
void main()
{
int ar1[5]={5,10,15,20,25};           // Array declaration statement
int a,b, ar2[5];
```

```
cout<<"\n"<<"The array elements are:"<<endl;
for(a=0;a<5;a++)
{
cout<<"    "<<ar1[a];
}
        // Copying elements into array2 starting from end of array.
for(a=5-1,b=0;a>=0;a--,b++)
{
ar2[b]=ar1[a];
}
        // Copying reverse array into original
for(a=0;a<5;a++)
{
ar1[a]=ar2[a];
}
cout<<"\n"<<"Reversed array is: "<<"\n";     //Accessing sorted array members
for(a=0;a<5;a++)
{
cout<<"\t"<<ar1[a];
}
getch();
}
```

Output:

The array elements are:

5 10 15 20 25

Reversed array is:

25 20 15 10 5

7.2.2 The complexity analysis of insertion sort is:

Worst Case Time Complexity : $O(n^2)$

Best Case Time Complexity : $O(n)$

Average Time Complexity : $O(n^2)$

Space Complexity : $O(1)$

8. MERGE SORT

8.1 Merge sort:

Refers to a kind of sort algorithm for rearranging a list of numbers or items (or any other data structure that can only be accessed sequentially e.g. file streams) into a specified order.

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

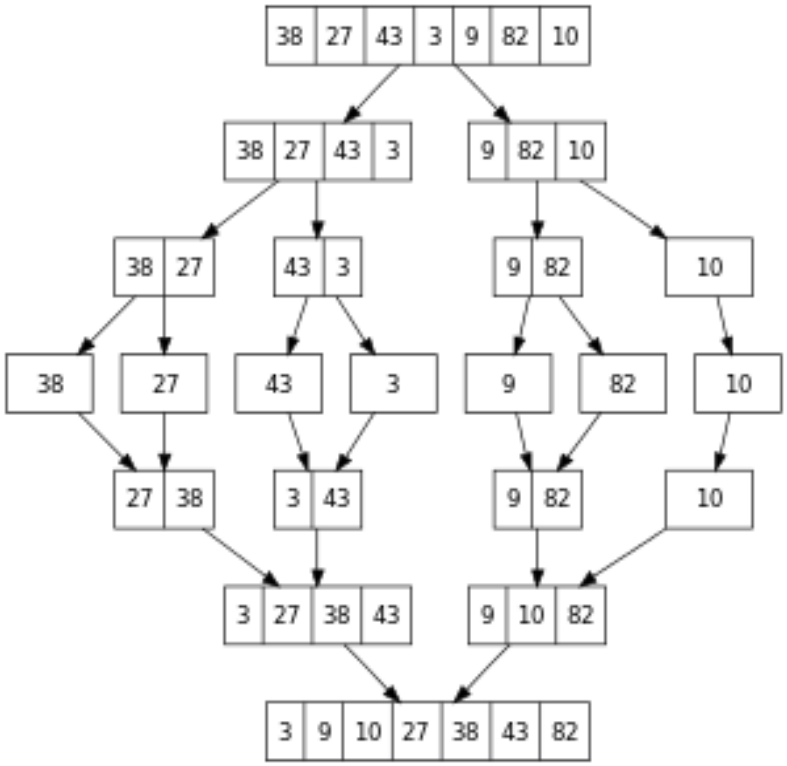

Figure 4: Shows the working of merge sort

8.2 Salient features:

- The algorithm was invented by John Von Neumann in 1945.
- It generally, divides the unsorted list into two sub lists of about half the size then sorts each of the two sub lists and merges the two sorted sub lists back into one sorted list.
- It is based on the divide and conquer paradigm.
- It is an O (n log n) comparison based sorting algorithm.

8.3 The process of merge sort algorithm is:

- The length of the list is 0 or 1,then it is considered as sorted.
- Otherwise, divide the sorted list into two (2) lists each about half the size.
- Sort each sub list recursively. Implement the step2 until the two sub lists are sorted.
- As a final step, we combine (merge) both the lists back into one sorted list.

The following program illustrates the implementation of the merge sort.

8.4 A program to sort array elements in ascending order using merge sort method:

```
#include<iostream.h>              //Function prototype
#include<conio.h>
#include<stdio.h>
void getdata(int ar[ ],int n)
{
int a;                          //variable declaration statement
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

```
cout<<"\n"<<"Entering array elements:"<<endl;
for(a=0;a<n;a++)
{
cin>>ar[a];
}
}
void display(int ar[ ],int n)
{
int a;
cout<<"\n"<<"Array elements are:"<<endl;
for(a=0;a<n;a++)
{
cout<<"   "<<ar[a];
}
getchar( );
}
void sort (int ar[ ],int low,int mid,int high)
{
Int i,j,k,l,ar1[20];
l=low;
i=low;
j=mid+1;
while((l<=mid)&&(j<=high))
{
if(ar[l]<=ar[j])
{
ar1[i]=ar[l];
l++;
}
else
{
ar1[i]=ar[j];
j++;
}
i++;
}
if(l>mid)
{
```

```
for(k=j;k<=high;k++)
{
ar1[i]=ar[k];
i++;
}
}
else
{
for(k=l;k<=mid;k++)
{
ar1[i]=ar[k];
i++;
}
}
for(k=low;k<=high;k++)
{
ar[k]=ar1[k];
}
}
void partition(int ar[ ],int low,int high)
{
int mid;
if(low<high)
{
mid=(low+high)/2;
partition(ar,low,mid);
partition(ar,mid+1,high);
sort(ar,low,mid,high);
}
}
void main()                    //Declaration of main function
{
int ar[20];
int n;
cout<<"\n"<<"Enter number of elements:"<<endl;
cin>>n;
getdata(ar,n);
partition(ar,0,n-1);
```

```
display(ar,n);
getchar();
}
```

Output:

```
Enter number of elements:
5
Entering array elements:
10
5
15
25
20
Array elements are:
5    10    15    20    25
```

8.5 The complexity analysis of merge sort is:

Worst Case Time Complexity : O(n log n)

Best Case Time Complexity : O(n log n)(typical), O(n) (natural variant)

Average Time Complexity : O(n log n)

Space Complexity : O(n)

9. SELECTION SORT

9.1 Selection sort: refers to a sorting technique basically used for sequencing a small list of items. It starts by comparing the entire list for the lowest item and moves it to the #1 position. It then compares the rest of the list for the next –lowest item and places it in the #2 position and so on until all item are in the require order.

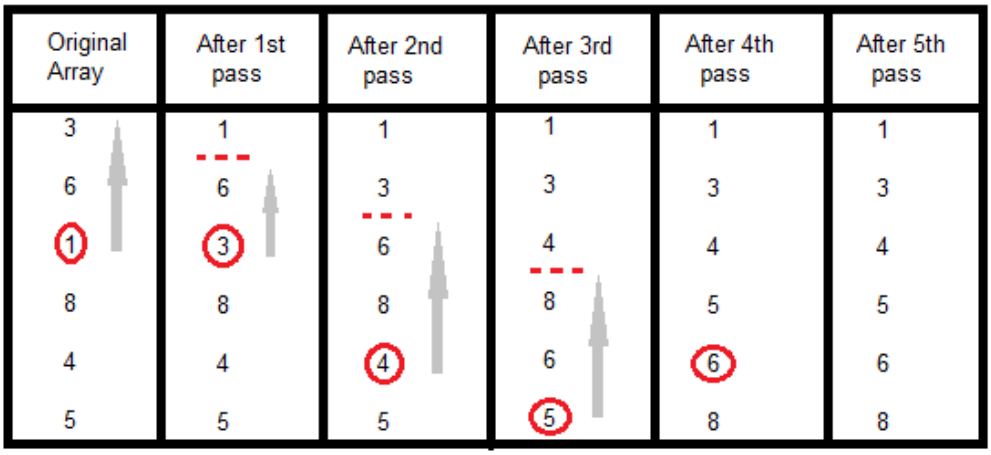

Figure 5: Shows the working of selection sort

9.2 Salient features:

- Selection sort is an in- place comparison sort.
- Selection sort is the most intuitive, simple and easiest to implement.

- It performs numerous comparisons, but fewer data movements than other methods.
- Selection sort shows good performance where auxiliary memory is limited.
- Selection sort is inefficient on large lists.

9.3 The process of selection sort algorithm is:

- Selection sort scan all elements/items and find the smallest. Then swap the smallest element into position as the first time.
- We repeat the selection sort on the remaining N-1 items/elements.
- Selection sort algorithm generally divides the input list into two parts: the sub-list of items already sorted, which is build up from left to right at the front(left) of the list, and the sub-list of items remaining to be sorted that occupy the rest of the list.

OR

- In the beginning, the sorted sub-list is empty and the unsorted sub-list is the entire input list.
- The selection sort algorithm proceeds by finding the smallest/largest (depending on sorting order) element in the unsorted sub-list, exchanging (swapping) it with the leftmost unsorted element (putting elements in sorted order) and moving the sub-list boundaries one element to the right.

OR

10.4 An algorithm for sorting an integer array using selection sort:

```
for( int i=0; i<ar.length -1;i++)
{
int min=i;
for(int j=i+1; j<ar.length; j++)
if(ar[j]<ar[min])
{
min=j;
}
int temp=ar[i];
ar[i]=ar[min];
ar[min]=temp;
}
```

The following program illustrates the implementation of the selection sort.

9.5 A program to sort array elements in ascending order using selection sort method:

```
#include<iostream.h>              //Function prototype
#include<conio.h>
void main()                  //Declaration of main function
{
int ar[5]={10,5,15,25,20};       // Array declaration statement
int a,b,n,position,swap;
n=5;
```

```
cout<<"\n"<<"Array members are:"<<endl;
for(a=0;a<n;a++)
{
cout<<"   "<<ar[a];
}
      //Initiating selection sort
for(a=0;a<(n-1);a++)
{
position=a;
for(b=a+1;b<n;b++)
{
if(ar[position]>ar[b])
{
position=b;
}
}
if(position!=a)
{
swap=ar[a];
ar[a]=ar[position];
ar[position]=swap;
}
}
cout<<"\n"<<"Sorted array elements are:"<<endl;
for(a=0;a<n;a++)
{
cout<<"   "<<ar[a];
}
getch();                    //freeze the screen until any key is pressed
}
```

Output:

Array members are:

10 5 15 25 20

Sorted array elements are:

5 10 15 20 25

9.6 The complexity analysis of selection sort is:

Worst Case Time Complexity : $O(n^2)$

Best Case Time Complexity : $O(n^2)$

Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

Average Time Complexity : $O(n^2)$

Space Complexity : O(1)

10. HEAP SORT

10.1 Heap sort: refers to a sorting algorithm that works by first organizing the data to be sorted into special type of binary tree called a heap.

OR

Binary heap----- refers to a heap data structure created using a binary tree. It canbe thought as a binary tree with two additional constraints:-

Shape property ------A binary heap is a complete binary tree, that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and if the last level of the tree is not complete, the nodes of that level are filled from left to right.

Heap property -------All nodes are either greater than or eq2ual to or less than or equal to each of its children, according to a comparison predicate defined for a heap.

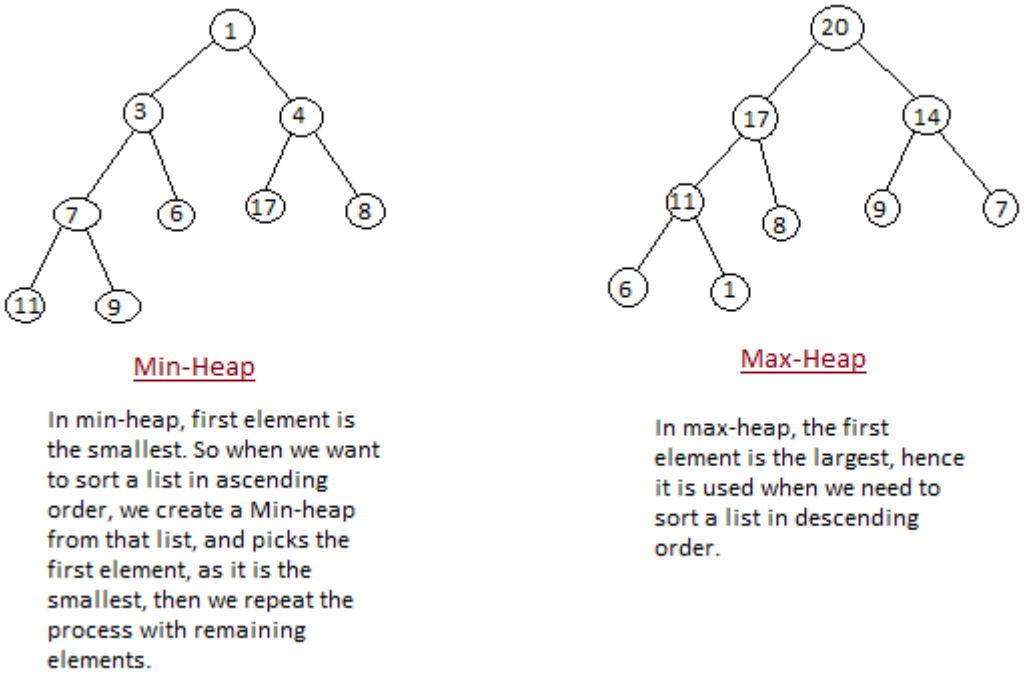

Figure 6: Shows the different types of heap

Consider {6, 5, 3, 1, 8, 7, 2, 4} to be the list that we want to sort from the smallest to the largest.

At first, we build the heap

Heap	newly added element	swap elements
nil	6	
6	5	
6, 5	3	
6, 5, 3	1	
6, 5, 3, 1	8	
6, **5**, 3, 1, **8**		5, 8

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

6, **8**, 3, 1, 5		6, 8
8, 6, 3, 1, 5	7	
8, 6, **3**, 1, 5, **7**		3, 7
8, 6, 7, 1, 5, 3	2	
8, 6, 7, 1, 5, 3, 2	4	
8, 6, 7, **1**, 5, 3, 2, **4**		1, 4
8, 6, 7, 4, 5, 3, 2, 1		

Secondly, we begin sorting

Heap	swap elements	delete element	sorted array	details
8, 6, 7, 4, 5, 3, 2, **1**	8, 1			swap 8 and 1 in order to delete 8 from heap
1, 6, 7, 4, 5, 3, 2, **8**		8		delete 8 from heap and add to sorted array
1, 6, **7**, 4, 5, 3, 2	1, 7		8	swap 1 and 7 as they are not in order in the heap
7, 6, **1**, 4, 5, **3**, 2	1, 3		8	swap 1 and 3 as they are not in order in the heap
7, 6, 3, 4, 5, 1, **2**	7, 2		8	swap 7 and 2 in order to delete 7 from heap
2, 6, 3, 4, 5, 1, **7**		7	8	delete 7 from heap and add to sorted array
2, **6**, 3, 4, 5, 1	2, 6		7, 8	swap 2 and 6 as they are not in order in the heap
6, **2**, 3, 4, **5**, 1	2, 5		7, 8	swap 2 and 5 as they are not in order in the heap
6, 5, 3, 4, 2, **1**	6, 1		7, 8	swap 6 and 1 in order to delete 6 from heap
1, 5, 3, 4, 2, **6**		6	7, 8	delete 6 from heap and add to sorted array
1, **5**, 3, 4, 2	1, 5		6, 7, 8	swap 1 and 5 as they are not in order in the heap
5, **1**, 3, **4**, 2	1, 4		6, 7, 8	swap 1 and 4 as they are not in order in the heap
5, 4, 3, 1, **2**	5, 2		6, 7, 8	swap 5 and 2 in order to delete 5 from heap
2, 4, 3, 1, **5**		5	6, 7, 8	delete 5 from heap and add to sorted array
2, **4**, 3, 1	2, 4		5, 6, 7, 8	swap 2 and 4 as they are not in order in the heap
4, 2, 3, **1**	4, 1		5, 6, 7, 8	swap 4 and 1 in order to delete 4 from heap
1, 2, 3, **4**		4	5, 6, 7, 8	delete 4 from heap and add to sorted array
1, 2, **3**	1, 3		4, 5, 6, 7, 8	swap 1 and 3 as they are not in order in the heap
3, 2, 1	3, 1		4, 5, 6, 7, 8	swap 3 and 1 in order to delete 3 from heap
1, 2, 3		3	4, 5, 6, 7, 8	delete 3 from heap and add to sorted array
1, 2	1, 2		3, 4, 5, 6, 7, 8	swap 1 and 2 as they are not in order in the heap
2, 1	2, 1		3, 4, 5, 6, 7, 8	swap 2 and 1 in order to delete 2 from heap
1, 2		2	3, 4, 5, 6, 7, 8	delete 2 from heap and add to sorted array
1		1	2, 3, 4, 5, 6, 7, 8	delete 1 from heap and add to sorted array
			1, 2, 3, 4, 5, 6, 7, 8	completed

Figure 7: The table shows the working of heap sort

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

10.2 Salient features:

- Heap sort is invented by J.R.J Williams.
- Heap sort is a comparison based sorting algorithm to create a sorted array or list.
- Heap sort requires only a constant space for sorting a list.
- In this case, the largest value is kept at the top of the tree, so the heap sort algorithm must also reverse the order.
- Heap sort is very fast and is widely used for sorting.
- Heap sort is not a stable sort.

10.3 The process of heap sort algorithm is:

Heap sort algorithm is divided into two basic parts:

- At first, a heap of the unsorted array or list is created.
- In the second**,** a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal. Once all objects have been removed from the heap, we have a final/complete sorted array.

The following program illustrates the implementation of the heap sort.

10.4 A program to sort array elements in ascending order using heap sort method

```
#include<iostream.h>          //Function prototype
#include<conio.h>
void up(int *,int);
void down(int *,int,int);
void main()                   //Declaration of main function
{
int ar[5],a,b,c,n;            //Array and variable declaration statement
n=5;
cout<<"\n"<<"Enter array elements:"<<endl;
for(a=1;a<=n;a++)
{
cin>>ar[a];
up(ar,a);
}
b=n;
for(a=1;a<=b;a++)
{
int temp;
temp=ar[1];
ar[1]=ar[n];
ar[n]=temp;
n- -;
```

```
down (ar,1,n);
}
n=b;
cout<<"\n"<<"Sorted array elements are:"<<endl;
for(a=1;a<=n;a++)
{
cout<<"   "<<ar[a];
}
getch();                    //freeze the screen until any key is pressed
}
void up(int *ar,int a)
{
int v=ar[a];
while((a>1)&&(ar[a/2]<v))
{
ar[a]=ar[a/2];
a=a/2;
}
ar[a]=v;
}
void down(int *ar,int a,int n)
{
int v=ar[a];
int b=a*2;
while(b<=n)
{
If((b<n)&&(ar[b]<ar[b+1]))
b++;
if(ar[b]<ar[b/2])
break;
ar[b/2]=ar[b];
b=b*2;
}
ar[b/2]=v;
}
```

Output:

Enter array elements:

10

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

5

15

25

20

Sorted array elements are:

5 10 15 20 25

10.5 The complexity analysis of heap sort is:

Worst Case Time Complexity : O(n log n)

Best Case Time Complexity : O(n log n)

Average Time Complexity : O(n log n)

Space Complexity : O(n)

11. RADIX SORT

11.1 Radix sort ---- refers to a non comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value.

Consider the following example.

Original, unsorted list:

170, 45, 75, 90, 802, 2, 24, 66

Sorting by least significant digit (1s place) gives:

170, 90, 802, 2, 24, 45, 75, 66

Notice that we keep 802 before 2, because 802 occurred before 2 in the original list, and similarly for pairs 170 & 90 and 45 & 75.

Sorting by next digit (10s place) gives:

802, 2, 24, 45, 66, 170, 75, 90

Notice that 802 again comes before 2 as 802 comes before 2 in the previous list.

Sorting by most significant digit (100s place) gives:

2, 24, 45, 66, 75, 90, 170, 802

It is important to realize that each of the above steps requires just a single pass over the data, since each item can be placed in its correct bucket without having to be compared with other items.

11.2 Salient features:

- It is considered as a fast and stable sorting algorithm which can be used to sort data items that are identified by unique keys.
- Radix sort is a small method generally used when we want to alphabetize a large list of names .As for as sorting of numbers is concerned, radix sort do counter intuitively by sorting on the least significant digits first on the first pass integer numbers sorts on the least significant digit and combine in a array. Then on the second pass, the integer numbers are sorted again on the second least significant digits and combine in an array and so on.

11.3 The process of radix sort algorithm is:

- For each digit i where i varies from least significant digit (LSD) to the most significant digit (MSD).

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

- We sort input array using counting sort (or any stable sort) according to the i^{th} digit.

The following program illustrates the implementation of the radix sort.

11.4 A program to sort array elements in ascending order using radix sort method:

```
#include<iostream.h>                    //Function prototype
#include<conio.h>
#include<stdio.h>
#define max 100
#define showpass
void display(int *ar1, int n)
{
int a;
for(a=0;a<n;a++)
{
cout<<"   "<<ar1[a];
}
}
void radixsort(int *ar1, int n)
{
int a,ar2[max],m=0,exp=1;
for(a=0;a<n;a++)
{
if(ar1[a]>m)
m=ar1[a];
}
while(m/exp>0)
{
int box[10]={0};
for(a=0;a<n;a++)
box[ar1[a]/exp%10]++;
}
for(a=1;a<10;a++)
box[a]+=box[a-1];
for(a=n-1;a>=0;a--)
ar2[--box[ar1[a]/exp%10]]=ar1[a];
for(a=0;a<n;a++)
```

```
ar1[a]=ar2[a];
exp*=10;
#ifdef showpass
cout<<"\n"<<"Pass:";
display(ar1,n);
#endif
}
}
void main()                    //Declaration of main function
{
int ar3[max];
int a,n1;
clrscr();
cout<<"\n"<<"Enter total elements:";
cin>>n1;
cout<<"\n"<<"Enter "<<n1<<"array elements:"<<endl;
for(a=0;a<n1;a++)
{
cin>>ar3[a];
}
cout<<"\n"<<"Array:";
display(&ar3[0],n1);
radixsort(&ar3[0],n1);
cout<<"\n"<<"Sorted:";
display(&ar3[0],n1);
getch();                    //freeze the screen until any key is pressed
}
```

Output:

```
Enter total elements: 3
Enter 3 elements:
5
15
10
Array:    5     15     10
Pass:     10    5      15
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

Pass: 5 10 15

Sorted: 5 10 15

11.5 The complexity analysis of radix sort is:

Worst Case Time Complexity : O(n +k)

Best Case Time Complexity : O(n+k)

Average Time Complexity : O(n +k)

Space Complexity : O(n+k)

12. SHELL SORT

12.1 Shell sort ----- refers to the first diminishing increment sort.

The following example shows the working of shell sort.

Let 3 7 9 0 5 1 6 8 4 2 0 6 1 5 7 3 4 9 8 2 be the data sequence to be sorted. First, it is arranged in an array with 7 columns (left), then the columns are sorted (right):

3 7 9 0 5 1 6		3 3 2 0 5 1 5
8 4 2 0 6 1 5	□	7 4 4 0 6 1 6
7 3 4 9 8 2		8 7 9 9 8 2

Data elements 8 and 9 have now already come to the end of the sequence, but a small element (2) is also still there. In the next step, the sequence is arranged in 3 columns, which are again sorted:

3 3 2		0 0 1
0 5 1		1 2 2
5 7 4		3 3 4
4 0 6	□	4 5 6
1 6 8		5 6 8
7 9 9		7 7 9
8 2		8 9

Now the sequence is almost completely sorted. When arranging it in one column in the last step, it is only a 6, an 8 and a 9 that have to moved a little bit to their correct positions.

12.2 Salient features:

- Shell sort was invented in 1959 by Donald L.Shell.
- In this case, on each pass i sets of n/I items sorted, typically with insertion sort.
- Shell sort is one of the oldest sorting algorithm.
- Shell sort is fast and easy to implement.
- Shell sort is considered to be an extension of insertion sort.
- Shell sort is not stable.

12.3 The process of shell sort algorithm is:

h=1

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 1, pp: (171-199), Month: January-April 2016, Available at: www.noveltyjournals.com

while h<n, h=3*h+1

while h>0

h=h/3

for k=1:k; insertion sort a[k:h:n] → invariant: each h sub array is sorted

end

The following program illustrates the implementation of the shell sort.

12.4 A program to sort array members/elements in ascending order using shell sort method:

```
#include<iostream.h>              //Function prototype
#include<conio.h>
#define max 5
void shellsort(int ar[ ],int tot,int index);
void main(void)                   //Declaration of main function
{
int ar[max]={0};
int a=0;
clrscr();
cout<<"\n"<<"Enter array elements:"<<endl;
for(a=0;a<max;a++)
{
cin>>ar[a];
}
cout<<"\n"<<"Array members are:"<<endl;
for(a=0;a<max;a++)
{
cout<<"   "<<ar[a];
}
cout<<"\n";
shellsort(ar,max,1);
cout<<"\n"<<"Sorted array elements are:"<<endl;
for(a=0;a<max;a++)
{
cout<<"   "<<ar[a];
}
getch();                  //freeze the screen until any key is pressed
}
```

```
void shellsort(int ar1[ ],int tot,int index)
{
int i =0;
int j=0;
int k=0;
int l=0;
for(k=0;k<index;k++)
{
for(i=k;i<tot;i+=index)
{
l=ar1[i];
for(j=(i-index);j>=0;j-=index)
{
if(ar1[j]>l)
{
ar1[j+index]=ar1[j];
}
else
{
break;
}
}
ar1[j+index]=l;
}
}
return;
}
```

Output:

Enter array elements:

10

5

15

25

20

Array members are:

10 5 15 25 20

Sorted array elements are:

5 10 15 20 25

12.5 The complexity analysis of shell sort is:

Worst Case Time Complexity : $O(n^2)$

Best Case Time Complexity : O(n)

Average Time Complexity : O(n log n)

Space Complexity : O(1)

13. CONCLUSION

The paper is intended to define, describe, understand and discuss some well known and useful sorting techniques most often used by computer programmers using C++. The paper also discusses the key features and characteristics of different sorting techniques along with suitable example of each type.

ACKNOWLEDGEMENT

I express my deep gratitude to my students and people around me who most often raise some computer programming related issues/problems that really inspired and motivated me to undertake this research work and make the things simple and precise. In the end, I would like to thank the great almighty who has given wisdom, strength and knowledge to visualise and explore things from grass root level and put on papers for the benefit of mankind.

REFERENCES

[1] Donald Knuth,The art of computer programming, volume 3: sorting and searching , Third Edition . Addison-Wesley, 1997. ISBN 0-201-89685-0.

[2] Robert Sedgewick. Algorithms in C++, Part 1-4 Fundamentals, Data structures, sorting, Searching: Pts 1-4, Second Edition. Addison Wesley Longman, 1998, ISBN 0-201-35088-2 Pages 273-274.

[3] Anany Levitin. Introduction to the Design & Analysis of Algorithms, 2nd Edition. ISBN 0-321-35828-7- Section 3.1: Selection Sort, PP98-100.

[4] [Sh] D.L.Shell: A high speed sorting procedure communications of the ACM2(7) , 30-32(1959).

[5] Dictionary of Algorithms and Data Structures: Shell Sort (http://www.nist.gov/dads/HTML/Shell Sort.html)

[6] Paul E.Black , ed , U.S.National Institute of Standards and Technology.22 June 2012.

[7] A Gibbons and W.Rytter, “Efficient parallel Algorithms” Cambridge university Press, 1988.

[8] Article (http://www.codercorner.com/Radix sort Revisited.html) about Radix Sorting IEEE Floating point numbers with implementation.

[9] Cormen,Thomas H; Leiserson,Charles E; Rivest, Ronald L; Stein Clifford(2001) , “2.1:Insertion sort”, Introduction to Algorithms(Second ed) , MIL Press and Mc Graw Hill,PP-15-21 , ISBN 0-262-03293-7.

[10] Insertion sort (Illustrated explanation), Algo list, Java and C++ implementations.