Animesh Maiti

2130006

Assingment-3

*1.Bubble Sort*

Inventor-

Iverson in 1962

Graphical Process-

Pass-1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24 | 54 | 19 | 20 | 17 | No swap |
| 24 | 54 | 19 | 20 | 17 | swap |
| 24 | 19 | 54 | 20 | 17 | swap |
| 24 | 19 | 20 | 54 | 17 | swap |

Pass-2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24 | 19 | 20 | 17 | 54 | swap |
| 19 | 24 | 20 | 17 | 54 | swap |
| 19 | 20 | 24 | 17 | 54 | swap |
| 19 | 20 | 17 | 24 | 54 | No swap |

Pass-3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 19 | 20 | 17 | 24 | 54 | No swap |
| 19 | 20 | 17 | 24 | 54 | swap |
| 19 | 17 | 20 | 24 | 54 | No swap |
| 19 | 17 | 20 | 24 | 54 | No swap |

Pass-4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 19 | 17 | 20 | 24 | 54 | swap |
| 17 | 19 | 20 | 24 | 54 | No swap |
| 17 | 19 | 20 | 24 | 54 | No swap |
| 17 | 19 | 20 | 24 | 54 | No swap |

Function code-

void bubbleSort(int arr[], int n)

{

    int i, j, temp;

    for (i = 0; i < n - 1; i++)

    {

        for (j = 0; j < n - i - 1; j++)

        {

            if (arr[j] > arr[j + 1]) *// condition for swaping*

            {

                swap(&arr[j], &arr[j + 1]);

            }

        }

    }

}

**Advantages-**

One of the main advantages of a bubble sort is that it is a very simple algorithm to describe to a computer. There is only really one task to perform (compare two values and, if needed, swap them).

**Disadvantages-**

The main disadvantage is the amount of time it takes. It is highly inefficient for large data sets, with a running time of O(n2).

**Complexity of the sorting algorithm**

1. **Best Case Complexity**

**If already sorted O(n)**

1. **Average Case Complexity**

**O(/2)**

1. **Worst Case Complexity**

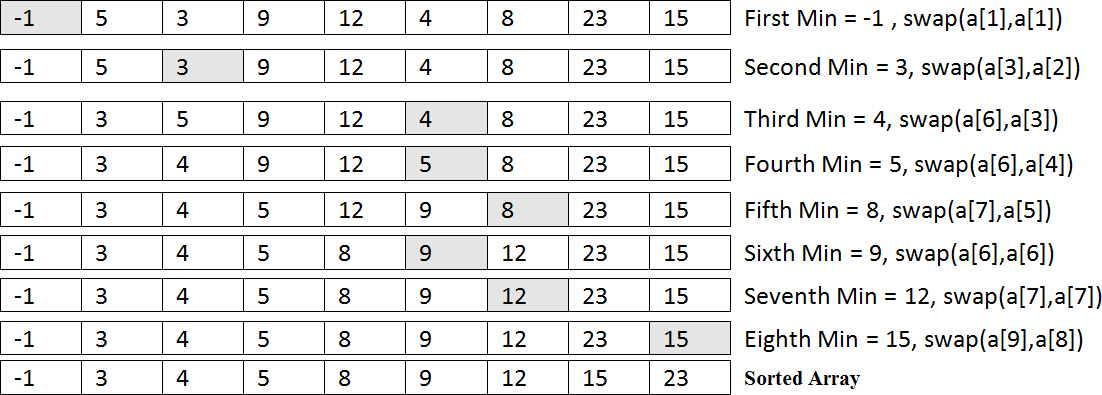
**O()**

*2.Selection Sort*

*Inventor-*

*Oscar Wilde*

Graphical Process-



Function code-

void selectionSort(int arr[], int n)

{

    int i, j, minIdx;

*// One by one move boundary of unsorted subarray*

    for (i = 0; i < n-1; i++)

    {

*// Find the minimum element in unsorted array*

        minIdx = i;

        for (j = i+1; j < n; j++)

          if (arr[j] < arr[minIdx])

            minIdx = j;

*// Swap the found minimum element with the first element*

           if(minIdx != i)

            swap(&arr[minIdx], &arr[i]);

    }

}

Advantages-

Selection sort algorithm is 60% more efficient than bubble sort.

the selection is based on keys and swaps only if necessary.

Disadvantages-

The selection sort's primary disadvantage is its inefficiency when the input size increases, and its performance decreases than the insertion sort algorithm.

Complexity-

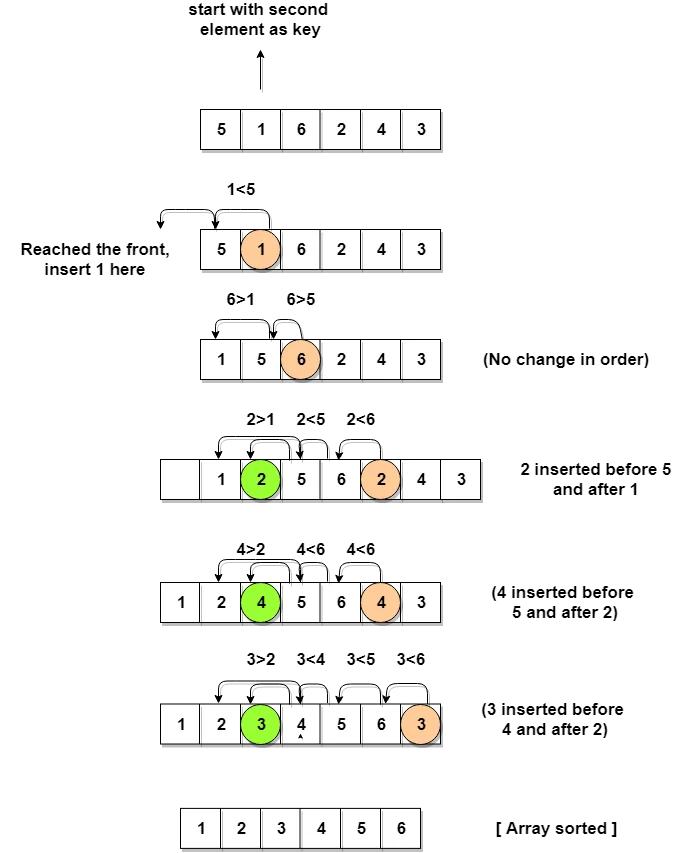
*O*(*n*2)

3.insertion sort

Inventor-

John Mauchly

Graphical process-



Function code-

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++) {

        key = arr[i];

        j = i - 1;

*/\* Move elements of arr[0..i-1], that are*

*greater than key, to one position ahead*

*of their current position \*/*

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

Advantages-

It also exhibits a good performance when dealing with a small list. The insertion sort is an in-place sorting algorithm so the space requirement is minimal.

Disadvantages-

Insertion sort is inefficient against more extensive data sets

Complexity-

Worst case- **O()**

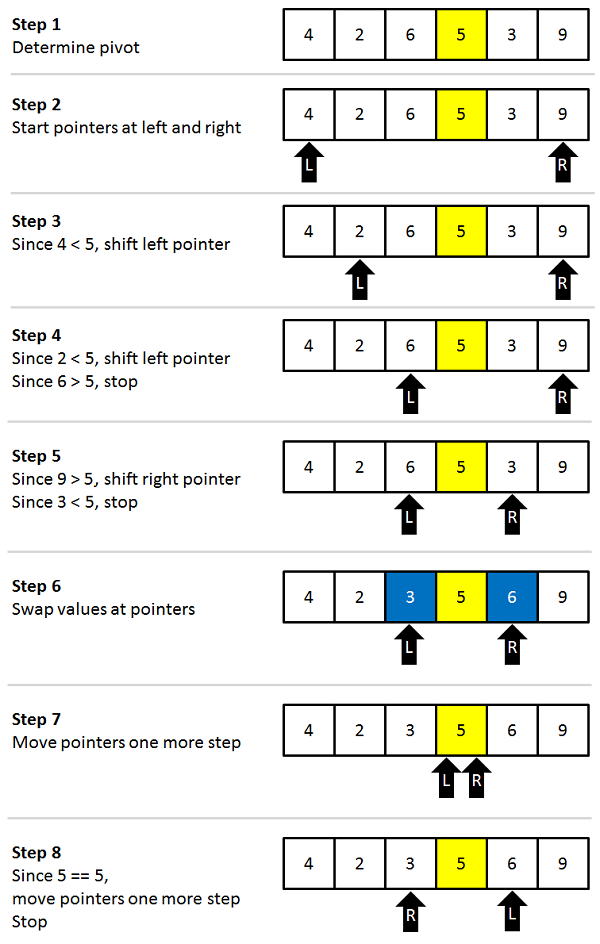
Best case- O(n)

4.Quick Sort-

Inventor-

[Tony Hoare](https://www.google.com/search?bih=520&biw=1280&hl=en&sxsrf=ALiCzsY76nLuUnuOmhmyb3hJ1OOgHF4_lg:1669820869380&q=Tony+Hoare&stick=H4sIAAAAAAAAAONgVuLQz9U3sCxJLn_EaMwt8PLHPWEprUlrTl5jVOHiCs7IL3fNK8ksqRQS42KDsnikuLjgmngWsXKF5OdVKnjkJxalAgA-n8tvTwAAAA&sa=X&ved=2ahUKEwigzIe-l9b7AhXnZWwGHXE3CMoQzIcDKAB6BAgLEAE)

Graphical process-



Function Code-

int partition(int arr[], int lb, int ub)

{

    int first = arr[lb];

    int start = lb, end = ub;

    while (start < end)

    {

        while (arr[start] <= first)

        {

            start++;

        }

        while (arr[end] > first)

        {

            end--;

        }

        if (start < end)

        {

            swap(&arr[start], &arr[end]);

        }

    }

    swap(&arr[lb],&arr[end]);

    return end;

}

int quickShort(int arr[],int lb,int ub){

    int loc;

    if (lb<ub)

    {

        loc=partition(arr,lb,ub);

        quickShort(arr,lb,loc-1);

        quickShort(arr,loc+1,ub);

    }

}

Advantages-

Sorting objects takes less time.

Disadvantages-

It is recursive. Especially, if recursion is not available, the implementation is extremely complicated.

Complexity-

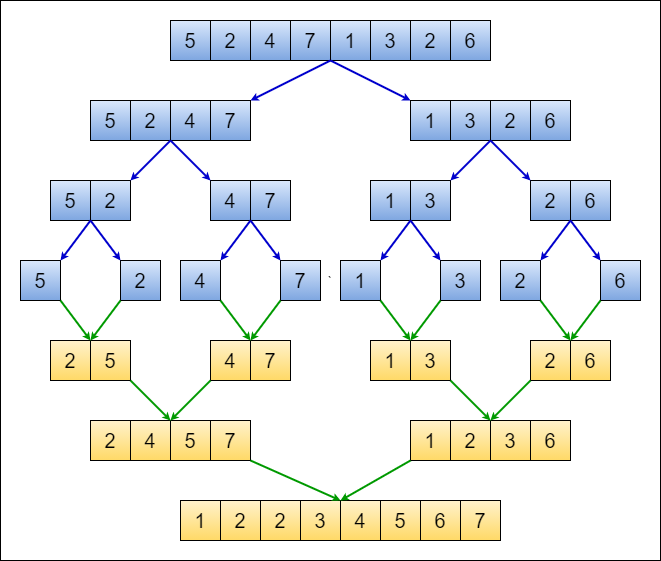
**O(n\*logn)**

5. Merge sort-

Inventor-

John Von Neumann

Graphical Process-



Function code-

int merge(int arr[], int lb, int mid, int ub)

{

    int i = lb, j = mid + 1, k = lb;

    int b[ub + 1];

    while (i <= mid && j <= ub)

    {

        if (arr[i] <= arr[j])

        {

            b[k] = arr[i];

            i++;

            k++;

        }

        else

        {

            b[k] = arr[j];

            j++, k++;

        }

    }

    if (i > mid)

    {

        while (j <= ub)

        {

            b[k] = arr[j];

            j++, k++;

        }

    }

    else

    {

        while (i <= mid)

        {

            b[k] = arr[i];

            i++, k++;

        }

    }

    for (int i = lb; i <= ub; i++)

    {

        arr[i] = b[i];

    }

}

int mergeShort(int arr[], int lb, int ub)

{

    int mid;

    if (lb < ub)

    {

        mid = (lb + ub) / 2;

        mergeShort(arr, lb, mid);

        mergeShort(arr, mid + 1, ub);

        merge(arr, lb, mid, ub);

    }

}

Advantages-

Merge sort can be used with linked lists without taking up any more space.

Disadvantages-

For small datasets, merge sort is slower than other sorting algorithms

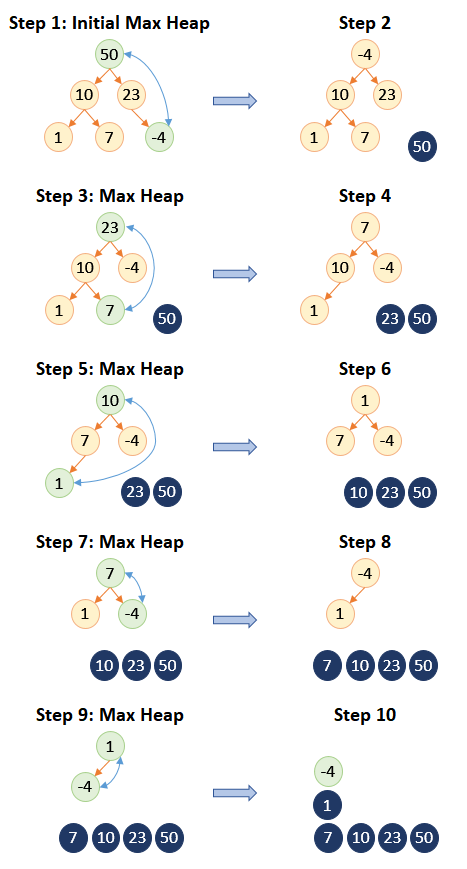
Complexity-

 O(n\*log(n))

6.Heap sort-

Inventor- **John William Joseph Williams**

Graphical process-



Function code-

void heapify(int arr[], int N, int i)

{

*// Find largest among root, left child and right child*

*// Initialize largest as root*

    int largest = i;

*// left = 2\*i + 1*

    int left = 2 \* i + 1;

*// right = 2\*i + 2*

    int right = 2 \* i + 2;

*// If left child is larger than root*

    if (left < N && arr[left] > arr[largest])

        largest = left;

*// If right child is larger than largest*

*// so far*

    if (right < N && arr[right] > arr[largest])

        largest = right;

*// Swap and continue heapifying if root is not largest*

*// If largest is not root*

    if (largest != i) {

        swap(&arr[i], &arr[largest]);

*// Recursively heapify the affected*

*// sub-tree*

        heapify(arr, N, largest);

    }

}

*// Main function to do heap sort*

void heapSort(int arr[], int N)

{

*// Build max heap*

    for (int i = N / 2 - 1; i >= 0; i--)

        heapify(arr, N, i);

*// Heap sort*

    for (int i = N - 1; i >= 0; i--) {

        swap(&arr[0], &arr[i]);

*// Heapify root element to get highest element at*

*// root again*

        heapify(arr, i, 0);

    }

}

Advantages-

Compared to quicksort, it has a better worst-case time complexity — **O(nlog n)**.  
The best-case complexity is the same for both quick sort and heap sort — **O(nlog n)**.

Disadvantages-

Heap Sort is considered unstable, expensive, and not very efficient when working with highly complex data.

Complexity-

Heap Sort has O(nlog n) time complexities for all the cases.

Complexity-

<https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/>

| **Algorithm** | **Time Complexity** | | | Space Complexity |
| --- | --- | --- | --- | --- |
|  | **Best** | **Average** | **Worst** | Worst |
| [Selection Sort](http://geeksquiz.com/selection-sort/) | Ω(n^2) | θ(n^2) | O(n^2) | O(1) |
| [Bubble Sort](http://geeksquiz.com/bubble-sort/) | Ω(n) | θ(n^2) | O(n^2) | O(1) |
| [Insertion Sort](http://geeksquiz.com/insertion-sort/) | Ω(n) | θ(n^2) | O(n^2) | O(1) |
| [Heap Sort](http://geeksquiz.com/heap-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) | O(1) |
| [Quick Sort](http://geeksquiz.com/quick-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(n) |
| [Merge Sort](http://geeksquiz.com/merge-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) | O(n) |
| [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/) | Ω(n +k) | θ(n +k) | O(n^2) | O(n) |
| [Radix Sort](https://www.geeksforgeeks.org/radix-sort/) | Ω(nk) | θ(nk) | O(nk) | O(n + k) |
| [Count Sort](https://www.geeksforgeeks.org/counting-sort/) | Ω(n +k) | θ(n +k) | O(n +k) | O(k) |
| [Shell Sort](https://www.geeksforgeeks.org/shellsort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(1) |
| [Tim Sort](https://www.geeksforgeeks.org/timsort/) | Ω(n) | θ(n log(n)) | O(n log (n)) | O(n) |
| [Tree Sort](https://www.geeksforgeeks.org/tree-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(n) |
| [Cube Sort](https://www.geeksforgeeks.org/sort-the-array-according-to-their-cubes-of-each-element/) | Ω(n) | θ(n log(n)) | O(n log(n)) | O(n) |