CS202

Data Structures and Algorithms

Assignment 1

Name- Aditya Malvi Roll no.- B18067

Report for sorting algorithms

Question1.

1) **Insertion Sort**

1) **Pseudo Code:**

```
1.for j=2 to n
2.      key=A[j]
3.      i=j-1
4.      while(A[i]>key and i>0)
5.              A[i+1]=A[i]
6.              i=i-1
7.      A[i+1]=key
```

2) **Time Complexity analysis:**

Best Case Running Time:O(n)

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

Average Case:$O(n^2)$

Space Complexity:O(1)

3) Remarks:

1) Good for sorting small arrays.

2) In place and stable

3) Not good for very large numbers as n^2 complexity.

2. Merge Sort

A)Pseudo Code:

```
1. Merge_Sort(A,p,r)
        2. If p<r
        3.      q=(p+r)/2
        4.      Merge_Sort(A,p,q)
        5.      Merge_Sort(A,q+1,r)
        6.      Merge(A,p,q,r)
        7. Merge(A,p,q,r)
        8.      i=p,j=q+1,k=0,tem[r-p+1]
        9.      while(i<q and j<=r)
        10.             if(A[i]<A[j)
        11.                     tem[k]=A[i]
        12.                     k+=1
        13.                     i+=1
        14.             else()
        15.                     tem[k]=A[i]
        16.                     j+=1
        17.                     k+=1
        18.              for(g=0 to n)
        19.                     A[g]=tem[g]
```

B) Complexity Analysis:

Best Case Running Time:O(nlogn)
Worst Case Running Time:O(nlogn)
Average Case:O(nlogn)
Space Complexity:O(n)

C) Remarks:

1) Not for large arrays as space complexity O(n).

2) Stable algorithm

3) Good point is it always works in O(nlogn) time complexity.

3. Quick Sort

A)Pseudo Code:

1) quickSort(A[],low,high)
2) if(low<high)
3) q=partition(A,low,high)
4) quickSort(A,low,q-1)
5) quickSort(A,q+1,high)
6) partition(A,low,high)
   a) pivot=A[high]
   b) i=low-1
   c) j=high+1
   d) while(true)
   e) j=j-1
   f) while(A[j]>pivot)
   g) i=i+1
   h) while(A[i]<pivot)
   i) if(j>i)
   j) exchange(A[i],A[j])
   k) Else if j=i
   l) Return j-1
   m) Else
   n) Return j

B) Complexity Analysis:

Best Case Running Time:O(nlogn)
Worst Case Running Time:O(n^2)
Average Case:O(nlogn)
Space Complexity:O(logn)

C) Remarks:

1) In place but not stable.
2) Better than merge sort as it takes less place
3) Can be time taking for large n.

3. Heap Sort

A)Pseudo Code:

1) HeapSort(A[])
2) BuildHeap(A)
3) For i= length downto 2
4) Swap A[i] and A[1]
5) heapsize-=1
6) Heapify(A,1,3)
7) BuildHeap(A)
8) heapsize=length(A)
9) For i=length/2 downto 1
10) Heapify(A,i)

Heapify(A,i)

1) left=2i+1
2) right=2i+2
3) If left<=heapsize and A[left]>A[i]
4) largest=left
5) Else
6) largest=i
7) If right<=heapsize and A[right]>A[i]
8) largest=right

9) Else

10) largest=i

11) If largest!=i

12) Exchange A[i] and A[largest]

13) Heapify(A,largest)

B) Complexity Analysis:

Best Case Running Time:O(nlogn)
Worst Case Running Time:O(nlogn)
Average Case:O(nlogn)
Space Complexity:O(1)

C) Remarks:

1) Always time complexity is nlogn.

2) In place but unstable

3) Memory efficient as no extra space required