Omkar R. Bhamitkar
112016020 ECE.

DSA Lab 9 : Theory

Q1. State and Explain : Sequential search and binary search. write detailed algorithm for the same.

→ ① A linear search also known as sequential search that simply scans each element at a time.

② A binary search is a search in which the middle element is calculated to check whether it is smaller or larger than the element which is to be searched.

Algorithm:

a) Sequential search.

```
Linear_Search (A, N, VAL, POS)

Step 1 : [initialize]   SET POS = -1
Step 2 : [initialize]   SET I = 0
Step 3 : Repeat step 4  while I < N
                        IF A[I] = VAL, then
```

SET POS = I
PRINT POS
Go to step 6.

[End of IF]
[End of loop]
step5: PRINT "Value not present in the array";
step6: Exit.

b) Binary Search

Binary-Search (A, lower-bound, upper-bound, VAL, POS)
step1: [initialize] set BEG = lower bound, End = upper bound, POS = -1
step2: Repeat step3 and step4 while BEG <= END.
step3: SET MID = (BEG + END)/2
step4: If A[MID] = VAL, then
POS = MID
PRINT POS
Goto step 6.

If A[MID] > Val then
SET END = MID - 1
Else
SET BEG = MID + 1
[End of IF]
[End of loop]
step5: If POS = -1, then
PRINT "VAL, IS not Present in the Array"

[END of IF]
Step 6: Exit.

Q2. State and Explain: Bubble Sort, Insertion Sort, Selection Sort, merge Sort, Heap Sort. and Quick Sort. Write detailed algo. for all sorting methods.

→ a) Bubble Sort:

Bubble sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements.

Algorithm:

Bubble_Sort (A, N)

Step 1: Repeat Step 2 for I = 0 to N-1
step 2: Repeat for J = 0 to N-1
step 3: IF A[J] > A[J+1], then
SWAP A[J] and A[J+1]
[End of Inner loop]
[End of Outer loop]
step 4: Exit.

b) Insertion Sort:

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hand.

Algorithm:

insertion sort (ARR, N) where ARR is an array of N elements

step 1: Repeat step 2 to 5 for k = 1 to N
step 2: Set temp = ARR[k]
step 3: Set J = k - 1
step 4: Repeat while temp <= ARR[J]
set ARR[J+1] = ARR[J]
set J = J - 1
[End of Inner Loop]
steps: set ARR[J+1] = temp
[End of loop]
step 6: Exit.

c) Selection Sort:

The Smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array

Algorithm:

Selection Sort to Sort an array ARR with N elements

Step 1: Repeat Step 2 and 3 for K = 1 to N-1
Step 2: Call Smallest (ARR, K, N, Pos)
Step 3: Swap A[K] with ARR[Pos]
[End of loop]
Step 4: Exit.

~~4)~~ ~~Merge~~ ~~Sort~~:

Smallest (ARR, K, N, Pos)

Step 1: [initialize] set Small = ARR[K]
Step 2: [initialize] set Pos = K
Step 3: Repeat for J = K+1 to N
IF SMALL > ARR[J], then
Set Small = ARR[J]
Set Pos = J
[End OF IF]
[End OF LOOP]
Step 4: Exit.

d) Merge Sort :

Merge Sort is a divide and conquer algorithm. It works by recursively breaking down a problem into two or more sub-problems of the same or ~~relative~~ related type, until these become simple enough to be solved directly.

Algorithm:

Merge_Sort (ARR, BEG, END)

Step 1: IF BEG < END, then
Set MID = (BEG + END)/2
Call Merge_Sort (ARR, BEG, MID)
Call Merge_Sort (ARR, MID + 1, END)
Merge (AR, BEG, MID, END)
[End of IF]

Step 2: End.

e) Heap Sort :

Heap Sort is a comparison-based sorting technique based on binary Heap data structure. It is similar to selection sort where we first find the minimum element and place the minimum element at the beginning.

Algorithm:

HeapSort (ARR, N)

Step 1: [Build Heap H]
Repeat for I = 0 to N-1
CALL Insert-Heap (ARR, N, ARR[
[End of Loop]

Step 2: [Repeatedly delete the root elemen
Repeat while N > 0
Call Delete-Heap (ARR, N, Val)
Set N = N+1
[End of loop]
Step 3: End.

F) Quick Sort:

Quick Sort is an in-place sorting algorithm. It is divided-and-conquer algo. It works by selecting a "pivat" element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivat.

Algorithm:

Quick_Sort (ARR, BEG, END)

Step 1: IF (BEG < END), then
Call Partition (ARR, BEG, END, LOC)
Call QuickSort (ARR, BEG, LOC - 1)
Call QuickSort (ARR, LOC + 1, END)
[End of IF]
Step 2: End.

Partition (ARR, BEG, END, LOC)

Step 1: [Initialize] Set Left = BEG; Right = END
LOC = BEG, Flag = 0
Step 2: Repeat Step 3 to while Flag = 0
Repeat while ARR[loc] <= ARR[Right]
and LOC != Right.
Set Right = Right - 1.
[End of loop]
IF LOC == Right; then
Set Flag = 1
Else IF ARR[LOC] > ARR[Right], then
Swap ARR[LOC] with ARR[Right]
Set LOC = Right
[End of IF]

IF flag = 0, then
Repeat while ARR[loc] >=
ARR[left] and loc != left
Set left = left + 1
[End of loop]
IF LOC == left, then
Set flag = 1
Else IF ARR[loc] < ARR[left], then
Swap ARR[LOC] with ARR[left]
Set loc = left.
[End of IF]
[End of IF]
Step 7 : [END of LOOP]
Step 8 : END.