ASSIGNMENT

By- Avi RAJ
CSE
11911037

1.) Insertion sort function

```
void insertion_Sort(int arr[], int n)
{
  for(int i = 1; i<n; i++) //----> O(n)
  {  j = i-1;
     key = Arr[i];
    while(j>=0 && Arr[j] > key) ---> O(n)
    {  A[j+1] = A[j];
       j--;
    }
    A[++j] = key;
  }
}
```

→ Time Complexity → Average/worst case = $O(n^2)$

→ Analysing Time complexity in best case
↳ Already sorted array

eg- Arr = | 2 | 3 | 4 | 5 | 6 |

the condition A[j] > A[i] is false everytime while loop doesn't run a single time.

So, only a for loop ⇒ O(n).

∴ Time Complexity for best case = O(n)

Technique to reduce time complexity. is to apply shell sort which uses insertion

sort on particular gaps and swaping indexes to reduce complexity to $O(n \log n)$

eg by taking gaps of 2.

eg - Arr

2	7	4	3	6
0	1	2	3	4

gaps = $4/2 = 2$

Insertion sorting at $i=0$ & $i=2$ $i=1$ & $i=3$

$i=2$ & $i=4$ & $i=0$ & $i=2$.

so, array looks like

2	3	4	7	6

now gap = $\frac{2}{2} = 1$, so it reduces the no. of swaping as it has been already done at a previous step and only swaps 7 with 6

arr = | 2 | 3 | 4 | 6 | 7 |

2) • Bubble Sort Algorithm -

• Adjacent elements are compared and swapped according to order.

5	3	4

Pass - I Comparing 5 with 3 $5 > 3$ swap

3	5	4

compare 5 with 4 $5 > 4$ swap

3	4	5

Pass - II Compare 3 with 4 $3 < 4$ No swap

compare 4 with 5 $4 < 5$ No swap

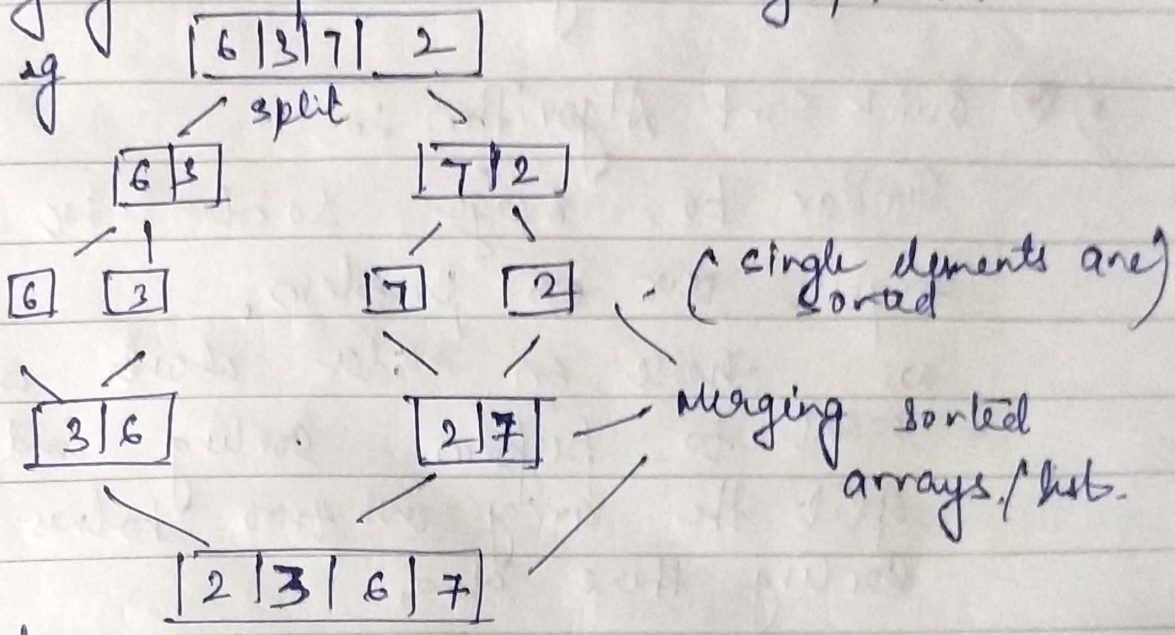
Time complexity ≃ $O(n^2)$, every element is compared to adjacent elements n times

space complexity ≃ $O(n)$, no extra space needed to sort, so "In Place Algorithm".

- Merge Sort Algorithm :-

↳ Merging two pre sorted arrays / list.

eg



Since at every step we are dividing in halfs and then copying & comparing items

time complexity ≃ $O(n \log n)$.

∴ Code :-

```
Void mergesort (int arr[], int l, int h)
{ if (l<h)
  { mid = (l+h)/2
    mergesort (arr, l, p)
    mergesort (arr, p+1, h)
    merge (arr, l, h) ; ⟶ O(n)
  }
```

$O(\log n)$

Space complexity → Merge sort requires extra space for sorting of at most n size and stacks of mergesort needs $\log n$ size (no. of calls).
Total size = $(n + \log n)$.

As extra space is required it is out of place Algorithm.

) Quick sort Algorithm :-
Similar to merge sort we divide & conquer our problem.
We choose an idea about which we want to perform sorting and then split the array in two halves and apply sorting there also.

eg - Arr = | 4 | 8 | 6 | 2 | 1 |

let sort about first element
so in pass I sorting about 4

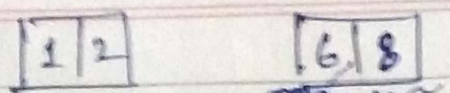
| 2 | 1 | 4 | 8 | 6 |

elements strictly smaller than 4 ↑ elements strictly greater than 4

split in two halves

| 2 | 1 | | 6 | 8 |

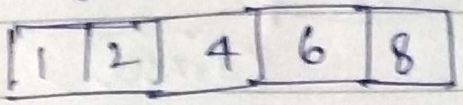
apply & sort about 1st element



element < 2

element > 6

now merge arrays back



sorted array.

Time complexity Average / best case = $O(n \log n)$

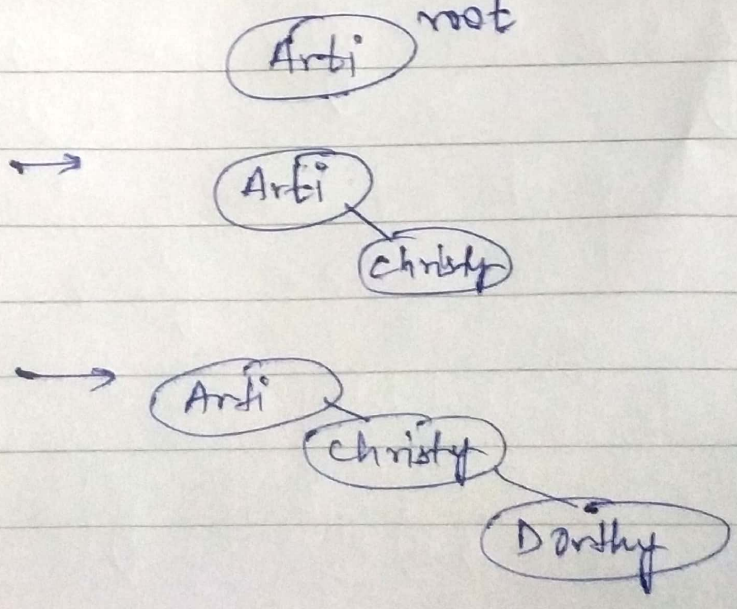
worst case = $O(n^2)$

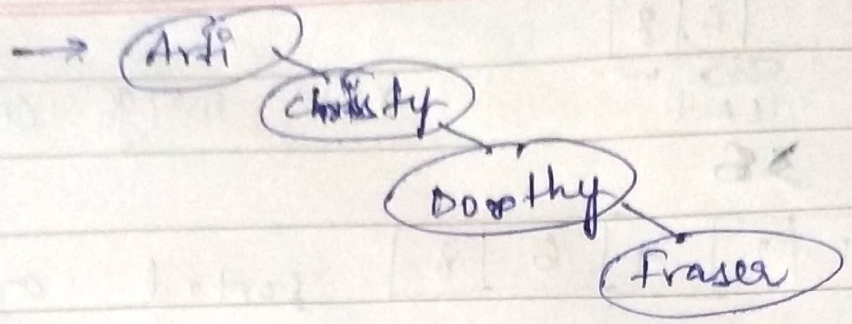
3.) Approach :-

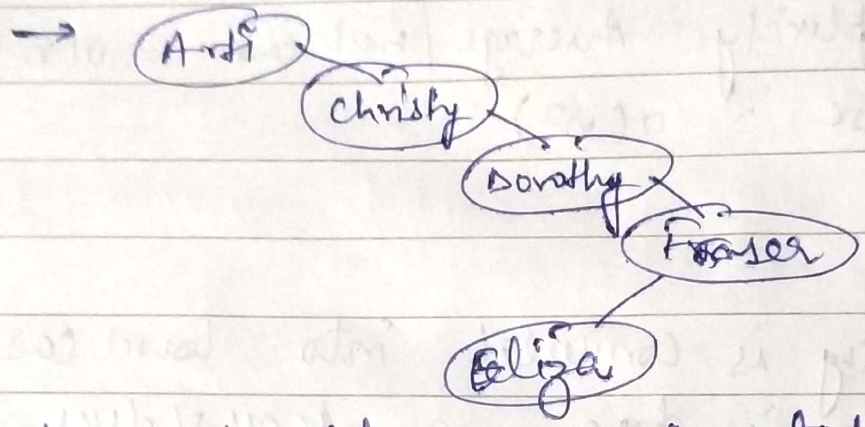
The string is converted into lower case & comparison is done on ASCII values, and is sorted through merge sort.

4.) Names are ~~done~~ taken in preorder and comparision is done on return values of strcmp (strcmp < 0 or strcmp > 0)

First element is made root.







Now David is inserted

Strcmp>0