CSE-203

30.3.20

Sorting

Sorting is arranging some elements either in ascending or descending order.

✱) Bubble Sort:

In each pass, compares the two adjacent numbers.
For n numbers, (n-1) passes

Time Complexity: $O(n^2)$

Two types of sorting:-

① In place → Collection of data fits into main computer memory
② External

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

(2)

CSE-203

31.3.20

*) Insertion Sort :-

Comparison starts from the second element, the previous elements are compared with the current element.

```
for j ← 2 to length[A]
    do key ← A[j]
       i ← j - 1
       while i > 0 and A[i] > key
           do A[i+1] ← A[i]
              i ← i - 1
           A[i+1] ← key
```

Correctness:

Loop invariant shows 3 things:-

① Initialization
② Maintenance
③ Termination

```
auto start = chrono::system_clock::now()
```

Counting time header:-

→ fopen returns a pointer
→ freopen directly takes i/o

```
freopen ("input.txt", "r", stdin);
int n;
cin >> n;
cout << n;
```

Time Complexity:

In each step, we do 1 step less starting from (n-1)

∴ Operation = $(n-1)+(n-2)+(n-3)+\ldots+3.2.1$

$= \frac{(n-1)(n)}{2}$ [summation upto (n-1)]

$= \frac{n^2-n}{2}$

∴ Θ Time complexity = $O(n^2)$

Worst case: $O(n^2)$

For Bubble sort: Best case: $O(n)$
Worst case: $O(n^2)$ → (can be optimized to $O(n)$)
Average case: $O(n^2)$

*) Selection sort:

An index is selected in every pass and then it is compared with the rest other elements.

```
for (int i=0; i<n-1; i++)
    for (int j=i+1; j<n; j++)
    {
        if (a[i] > a[j])
            swap (a[i], a[j]);
    }
```

Total pass: (n-1)
Total moves: 3 * (n-1)

Best/Worst/Average Case: $O(n^2)$

(4)

CW

CSE-203

1.4.

Mergesort

→ Recursive Algorithm, follows divide and conquer process.

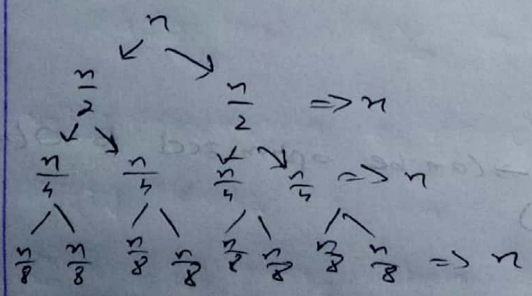
→ Divides list into halves and sorts them seperately.

Best / Worst / Average Case : $O(n \log n)$

→ Not an in-place sort

→ Good for sorting data with slow access times

Complexity → Until the array is divided into a single element



Total : $4n$ times

$n \times \log_2 n$ times

Upto $\frac{n}{2^k}$ $\quad \therefore \frac{n}{2^k} = 1$

$\Rightarrow 2^k = n$

$\Rightarrow k = \log n$

(5)

CW

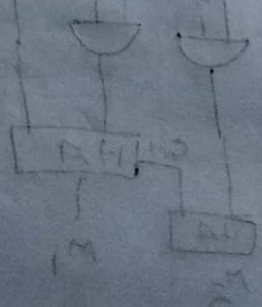
CSE-203

Quicksort

If one element, return
else
pick one element as pivot,
partition elements into two sub-arrays,
- elements less than or equal to pivot
- " greater than pivot

quicksort subarrays
return result

→ Divide and Conquer procedure
→ Quicksort is an in-place sort algorithm
→ First element is selected as pivot traditionally



CSE-203

& 5.5.20

Sortings

*) Selection Sort :

23 78 45 8 32 56

→ 8 78 45 23 32 56

→ 8 23 45 78 32 56

→ 8 23 32 45 78 56

→ 8 23 32 45 78 56

→ 8 23 32 45 56 78 [sorted]

*) Bubble Sort :

23 78 45 8 32 56

→ 23 78 45 8 32 56

→ 23 45 78 8 32 56

→ 23 45 8 78 32 56

→ 23 45 8 32 78 56

→ 23 45 8 32 56 78

→ 23 45 8 32 56 78

→ 23 8 45 32 56 78

→ 23 8 32 45 56 78

→ 23 8 32 45 56 78

→ 8 23 32 45 56 78

→ 8 23 32 45 56 78

→ 8 23 32 45 56 78 [sorted]

(7)

*) Insertion Sort :

23 78 45 8 32 56

→ 23 78 45 8 32 56

→ 23 45 78 8 32 56

→ 8 23 45 78 32 56

→ 8 23 32 45 78 56

→ 8 23 32 45 56 78 [sorted]

For Selection Sort,

Best Case = Worst Case = Average Case = $O(n^2)$

(8)

CSE-203

CW

7.5.20

*) Merge Sort:-

A sorting algorithm which follows divide-and-conquer proced

→ Recursive algorithm
→ Divides list into two halves
→ Sorts seperately
→ Merges together

6 3 9 1 5 4 7 2
→ 3 6 9 1 5 4 7 2
→ 3 6 1 9 5 4 7 2
→ 1 3 6 9 5 4 7 2
→ 1 3 6 9 4 5 7 2
→ 1 3 6 9 4 5 2 7
→ 1 3 6 9 2 4 5 7
→ 1 2 3 4 5 6 7 9

Best Case:

No. of moves = 2k
" " comparison = k

Worst Case:

No. of moves = 2k
" " comparison = 2k-1

⑨

CW

CSE-203

12.5.20

Searching Algorithm

The process of finding an element in an array is called searching.

2 types of searching: ① Linear Search
② Binary search

*) Linear Search:

Each member of the array is visited until the search key is found.

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cin>>n;
   int a[n];
   for(int i=0;i<n;i++)
       cin>>a[i];
   int key;
   cin>>key;
   int pos=-1;
   for(int i=0;i<n;i++)
   {
      if(a[i]==key)
      {
          pos=i+1;
          break;
      }
   }
```

```
if (pos==-1)
    cout<<"Value not found";
else
    cout<<"Value found at position "<<pos;
}
```

Worst Case : O(n) [Have to check all the elements if the value is absent or at the last of array]

*) Binary Search:

A sorted array is required for binary search. The searched array is divided by 2 for each comparison.

∴ Best Case = Worst Case = $O(\log_2 n)$

```
int Binary_search (int a[], int size, int key)
{
    int low=0, high=size-1, middle;
    while (low<=high)
    {
        middle = (low+high)/2;
        if (key == a[middle])
            return middle+1;
        else if (key < a[middle])
            high = middle-1;
        else
            low = middle+1;
    }
    return -1;
}
```

Note: This code is inconsistent. To get the first value, we can make a change

```
int low=0, high= size-1, mid;
int pos = -1;
while (low<=high)
{
   mid =(low+high) /2;
   if (key == a[middle])
   {
      high= pos = middle;
      high = middle-1;  → finds the very first identical value
   }
   ——  —  —
   ——  ——  ——
}
return pos;
```

or we can write
low = middle+1; → finds last value

*) Quicksort:

→ An in-place sort, no need for extra array.

6 3 9 1 5 4 7 2
→ 6 3 2 1 5 4 7 9
→ 4 3 2 1 5 6 7 9
→ 1 2 3 4 5 6 7 9

Best Case: $O(n \log_2 n)$

Best case will happen if the partition takes place at the middle.

Worst Case: $O(n^2)$

For already sorted array, split takes place at the start/end so for n element, split takes place n times.

VVI

Improved pivot selection of Quicksort