Searching & Sorting (Lec 20)

Linear Search (a, n, x)

a → array, n → size of array, x → element

(linsrch.c)

Binary Search (a, i, j, x)

a → array, i → start, j → end, x → element

(Recursive) (binsrch.c)

(Iterative) (binsrch2.c)

THEORY (Linear Search)

Time Complexity

→ BEST CASE: Key is present at first index [O(1)].

→ WORST CASE: Key is present at last index [O(N)].

→ AVG. CASE: O(N)

Advantages

→ Can be used irrespective of whether the array is sorted or not. Can be used on arrays of any data type.

→ Does NOT require any additional memory.

→ Well suited algo for small datasets.

Disadvantages

→ TC [O(N)] makes it slow for large datasets.
→ Not suitable for large arrays.

It is used when:

→ Dealing with a small dataset.
→ searching for a dataset stored in contiguous memory

THEORY (Binary Search)

TC

→ BEST CASE: O(1)
→ WORST CASE: O(log N)
→ AVG. CASE: O(log N)

Conditions to apply binary search:

→ data str. should be sorted.
→ access to any element of the data structure should take const. time

Advantages

→ Faster than linear search.
→ More efficient than other searching algos with a similar TC like interpolation search OR exponential search.

Disadvantages

→ Array should be sorted.

→ It requires the elements of array be comparable, they must be able to be ordered.

Insertion Sort

Simple sorting algo that works by iteratively inserting each element of an unsorted list into its correct posⁿ in sorted portion of the list.

Remember

```
void ins_sort(int arr[], int n) {

for(int i=1; i<n; i++){
    int key = arr[i];
    int j = i-1;

    while(j>=0 && arr[j]> key){
        arr[j+1] = arr[j];
        j = j-1;
    }
    arr[j+1] = key;
}
}
```

TC

→ BEST CASE : $O(n)$

→ WORST " : $O(n^2)$ → Randomly ordered

→ AVG. " : $O(n^2)$ → Reverse Order

SC (Auxillary Space : $O(1)$)

Advantages:

- Simple & easy
- efficient for small lists & nearly sorted lists

Disadvantages:

- Inefficient for large lists
- Not as efficient as other sorting algos like merge & quick sort.

Selection Sort (see diag - Lec 20)

Remember

```
void sel_sort(int arr[], int n){

  for(int i=0; i<n; i++){
      int min_idx = i;

      for(int j=i+1; j<n; j++){
          if(arr[j]< arr[min_idx]){
              min_idx = j;
          }
      }

      int temp = arr[i];
      arr[i] = arr[min_idx];
      arr[min_idx] = temp;
  }
}
```

TC

Overall complexity = $O(n^2)$

SC

Auxillary Space = $O(1)$

Advantages:

- → Easy to understand and implement
- → Requires only a constant $O(1)$ extra memory space.

Disadvantages:

- → TC of $O(n^2)$ makes it slower compared to algos like Quick Sort or Merge sort.
- → Does NOT maintain relative order of equal elements.

Bubble Sort (see diag - Lec 20)

Remember

```
void bubsort (int arr[], int n) {

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

TC $O(n^2)$
SC $O(1)$

Advantages

→ easy to understand & implement
→ Doesn't require any additional memory

Disadvantages

→ TC of $O(n^2)$ which makes it slow for large data sets.
→ Comparison based sorting algo limits efficiency of algorith[m] in certain cases.