Day 13

2/25/25 Sorting Algorithms

// Reviewed all previous notes on an assignment

Selection Sort — select min & swap

13	46	29	53	20	9
0	1	2	3	4	5

ascend → 9, 13, 20, 24, 46, 52

9	46	24	53	20	13	// Step 1
9	13	24	53	20	46	// Step 2
9	13	20	53	24	46	// Step 3
9	13	20	24	53	46	// Step 4
9	13	20	24	46	53	// Step 5

} every time get min & swap (algorithm)

swap happened at 0 index & min index [0-n-1]
next, swap happened at 1 index & min index [1-n-1]
continue till n-2 so... ↙

```
for(i=0; i<=n-1; i++){
  minValue = i;
  for(j=i+1; j<=n; j++){
    if(arr[j] < arr[minValue]) minValue = j;
  swap(arr[minValue], arr[i]);
```

arr[i] = 15, arr[minValue] = 12
temp = arr[minValue] (12)
arr[minValue] = arr[i] (15)
arr[i] = temp (12)
saved the 12 in temp

TC ≈ O(N²) → Best, Worst & Average

Bubble Sort

*pushes max to the back

13, 46, 24, 52, 20, 9
13, 24, 46, 52, 20, 9
13, 24, 46, 20, 52, 9
13, 24, 46, 20, 9, 52 // max is last but partial sort
13, 24, 46, 20, 9, 52
13, 24, 20, 46, 9, 52
13, 24, 20, 9, 46, 52 // 2nd max is 2nd last

*continue till sorted / condition met

```
for(i=n-1; i>=1; i--){  //going from last and i--

    for(j=0, j<i-1; j++){    //i-1 b/c last can compare to 2nd
      if(a[j] > a[j+1]) swap;  //comparing each one till swap
    }
}
```

TC ≈ $O(N^2)$ → Worst/Average Complexity

(*if no swaps best complexity and O(N)

```
didSwap = 0;
didSwap = 1; (inside of internal loop)

if (didSwap == 0){ } (in outer loop)
    break;
```

} best case scenario

Insertion Sort

*always takes an element and places it in correct position

9 14 15 12 6 8 13

9 14 15 12 6 8 13

9 14 15 12 6 8 13

final: 6 8 9 12 13 14 15

} checks increasing arr of given, then moves new addition to correct position from 0→n-1

```
for(i=0, i<=n-1; i++) { // run from 0 to end of array
    j=i-1;
    while(j>0 && array[j-1]>array[j]){ // if left smaller, swap
        swap(a[j-1], a[j]);
        j--;                          // once not smaller go to next
    }
}
```

TC = $O(N^2)$ Average / Worst

Best $O(N)$ when already sorted

Merge Sort

*takes better time complexity (divide & merge)

[3,1,2,4,1,5,2,6,4]

divide

[3,1,2,4,1] [5,2,6,4]

[3,1,2] [4,1]

[1,3] ← [3,1] [2]

[3] [1]

[1,3] [2]

[1,2,3]

} repeats divide & merge till all splits recombine to be sorted

arr = 5

low (start) ↑ high (end)

```
mergeSort(arr, low, high) {
→ low = 0;
  high = arr.size();
  mid = (low + high)/2;
  mergeSort(arr, low, mid);      // send to get sorted seperately
  mergeSort(arr, mid+1, high);
    merge(arr, low, mid, high)
[ if(low >= high) return;
```

Day 14

2/26/25

Quick Sort

* sorts in ascending and descending, slightly better than merge sort for time complexity

pivot = first, last, med, random, basically any element rule

pick up, place in correct

4 6 2 5 7 9 1 3 // individually select element and put to spot

1 2 3 4 5 6 7 9

2 1 3 | 4 | 6 5 7 9) final sort

smaller, set a/r, greater

2 1 3) recursion

1 2 3

small, large

6 5 7 9

5 6 7 9

small, large

7 9

7 9

large

finding low and high, but not creating new arrays, instead using pointers

low(i), high(j)

index → 0 1 2 3 4 5 6 7 8 9

unsorted array a[]

pivot = a[low];

```
quickSort(arr, low, high){
  if(low < high){
    pIndex = f(arr, low, high);
    quickSort(arr, low, pIndex-1);
    quickSort(arr, pIndex+1, high);
  }
}
```

```
int f(arr, low, high){
  pivot = arr[low];
  i = low;
  j = high;
  while(i<j){
  while(arr[i]<= pivot
      && i<=high){
  } i++;
  while(arr[i]> pivot
      && j> low){
  } j++;
  if(i<j) swap arr[i],
      arr[j];
```