## Selection Sort

- we select the smallest element in the entire list & swap.
- the first element gets placed after the first iteration.

→ iteration 1 : 0 – n
→ iteration 2 : 1 – n
→ iteration 3 : 2 - n

```
void selectionsort(int arr[], int n){
    for(int i=0; i<n-1; i++){
        int mini=i;
        for(int j=i+1; j<n; j++){
            if(arr[j] < arr[mini]){
                mini=j;
            }
        }
        if(mini != i) swap(arr[i], arr[mini]);
    }
}
```

Time Complexity : $O(\frac{n \times (n+1)}{2}) \sim O(n^2)$ ← $n + (n-1) + (n-2) \dots 1$

Best Case → $O(n^2)$
∵ even if you not swap, you still iterate
Worst Case → $O(n^2)$

## Bubble Sort

- we swap the adjacent elements if they're not sorted.
- last element gets in place after each iteration

```
void bubblesort(int arr[], int n){
    for(int i=0; i<n-1; i++){
        int flag=0;
        for(int j=0; j<n-i-1; j++){
            if(arr[j] > arr[j+1]){
                swap(arr[j], arr[j+1]);
                flag=1;
            }
        }
        if(flag==0) break; // already sorted
        cout << "runs\n"; // check
    }
}
```

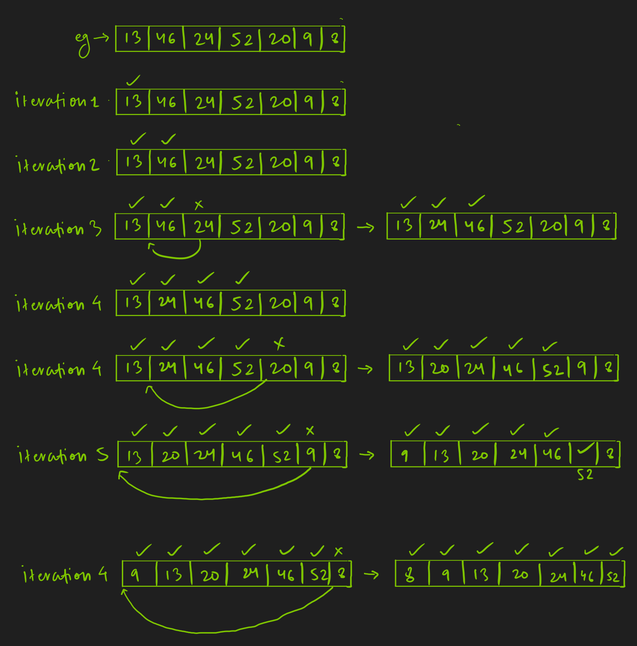
Time Complexity : $O(\frac{n \times (n-1)}{2}) \sim O(n^2)$
Best case : O(n) // sorted already
Worst case : $O(n^2)$ → descending

## Insertion Sort  Merge Sort

- select an element and place it in its correct position.

```
void insertionsort(int arr[], int n){
    for(int i=0; i<n; i++){
        int j=i;
        while(j>0 && arr[j-1] > arr[j]){
            swap(arr[j], arr[j-1]);
            j--;
        }
    }
}
```

Time Complexity : $O(\frac{n \times (n-1)}{2}) \sim O(n^2)$
Best case : O(n) // sorted already
Worst case : $O(n^2)$ → descending




## Merge Sort

```
void merge(vector<int> &arr, int low, int mid, int high){
    vector<int> temp;
    int left=low;
    int right=mid+1;
    while(left<=mid && right<=high){
        if(arr[left] <= arr[right]){
            temp.push_back(arr[left]);
            left++;
        }
        else{
            temp.push_back(arr[right]);
            right++;
        }
    }
    while(left<=mid){
        temp.push_back(arr[left]);
        left++;
    }
    while(right<=high){
        temp.push_back(arr[right]);
        right++;
    }
    for(int i=0; i<arr.size(); i++){
        arr[i]=temp[i-low];
    }
}

void mergesort(vector<int> &arr, int low, int high){
    if(low>=high) return.
    int mid=(low+high)/2;
    mergesort(arr, low, mid);
    mergesort(arr, mid+1, high);   } log N
    merge(arr, low, mid, high);    } O(N)
}
```

Time Complexity → O(N log N); ← merge()
Space Complexity → O(N) + O(N) ~ O(N)
auxiliary ↙  ↘ storing temp