CONTENT



# Problem 1

## 1-4.

Use default_random_engine and uniform_int_distribution from <random> to generate a random array. And recursively divide the array, sort the sub-arrays and merge the sub-arrays.

### Sample run.

```
Input the value of n(1<n<=50)> 51
n should in range (1,50]
```

```
Input the value of n(1<n<=50)> 25
Before merge sort, the random array A is:
0 13 76 46 53 22 4 68 68 94 38 52 83 3 5 53 67 0 38 6 42 69 59 93 85
After merge sort, the ordered array A is:
0 0 3 4 5 6 13 22 38 38 42 46 52 53 53 59 67 68 68 69 76 83 85 93 94
```

### 5.

When calling merge function, If sub-arrays are already sorted, return.

```
// if whole array is already sorted, skip
if (array[middle] < array[middle + 1])
{
    cout << "array[" << lower << ":" << upper << "] already sorted" << endl;
}
```

#### Sample run.

```
Before modified merge sort, the sorted array A is:
0 4 13 22 46 53 68 68 76 94
array[0:1] already sorted
array[0:2] already sorted
array[3:4] already sorted
array[0:4] already sorted
array[5:6] already sorted
array[8:9] already sorted
array[5:9] already sorted
array[0:9] already sorted
After modified merge sort, array A is:
0 4 13 22 46 53 68 68 76 94
```

## Problem 2

In order to remove the infinite value, we could check whether the subarray is ampty.

### Pseudo code

MERGE(*A*,*p*,*q*,*r*)

**If** *A*[*q*] <= *A*[*q*+1]<br>
    return<br>
**else**<br>
    $n_1$=*q*-*p*+1<br>
    $n_2$=*r*-*q*<br>
    Let *L*[1.. $n_1$+1] and *R*[1.. $n_2$+1] be new arrays<br>
    **for** *i* = 1 **to** $n_1$<br>
        *L*[ *i* ] = *A*[ *p*+*i*-1 ]<br>
    **for** *j* = 1 **to** $n_2$

```
        R[ j ] = A[ q+j ]
    i = 1
    j = 1
    for k = p to r
        if i > n_1
            A[k] = R[ j++ ]
        else if j > n_2
            A[k] = L[ i++ ]
        else if L[ i ] <= R[ j ]
            A[k] = L[ i ]
            i = i +1
        else
            A[k] = R[ j ]
            j = j + 1
```

```
MERGE-SORT(A,p,r)
1   If p<r
2       q = [(p+r)/2]
3       MERGE-SORT(A,p,q)
4       MERGE-SORT(A,q+1,r)
5       MERGE(A,p,q,r)
```

# Problem 3

## a.

Both F1 and F2 are calculating the nth power of 2.

## b.

F2 is faster

```
When n=30:
F1 time = 4.76545s
F2 time = 1e-06s
```

## c.

F1 is O(n): n times recursion and 1 add for each recursion)
F2 is O($log\ n$): do recursion only once for n/2, the time complexity is $log_2 n$
So that is why F2 is faster than F1

# Problem 4

## a

ProcedureX try to sort A with ascending order. First i is the start index of the array, for each i loop, j loops move the smallest element to A[i], and then i moves to the next index. (37 words)

## b

The worst case is for each i loop the program have to move the element at the end of the array.

So total operation is $(n-1)+(n-2)+\cdots+1=\frac{n^2}{2}$

The time complexity is O($n^2$)

# Problem 5

## pseudo code

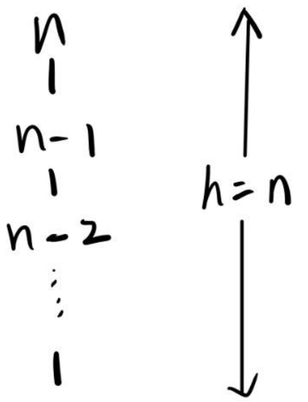
```
INSERTION-SORT(A,n)
    If n>1
        INSERTION-SORT(A,n-1)
        INSERTION(A,n)
```

## analysis

The running time of recursion insertionSort is n-1 and n for insertion. So the recurrence equation for insertion sort is:

$$T(n)=\begin{cases}1 & if\ n=2\\ T(n-1)+n & if\ n>2\end{cases}$$

Solve T(n)=T(n-1)+n




we need n times of insertion sort. Worst - case running time is ((n+1)×n)/2+n= Θ($n^2$).