P1 - Draw comparisons Between the sorting Algorithm Bubble Sort, Selection Sort , Insertion Sort ,Merge
Sort and Heap Sort ?

	Stable	**Time Complexity (Best Case)**	**Time Complexity (Average Case)**	**Time Complexity (Worst Case)**	**Inplace**
Bubble sort	Yes	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
Selection sort	No	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
Insertion sort	Yes	O(n) - (sorted)	$O(n^2)$	$O(n^2)$	Yes
Merge sort	Yes	O(n*logn)	O(n*logn)	O(n*logn)	No
Heap sort	No	O(n * log n)	O(n * log n)	O(n * log n)	Yes

References:

1. https://en.wikipedia.org/wiki/Heapsort

P2- Write a program to find out factorial for larger value like (100,150 etc.)?Note: No use of library

```
#include <iostream>
#include <vector>

using namespace std;

void multiply(vector<int>& fact, int n) {
    int carry = 0;
    for (int i = 0; i < fact.size(); i++) {
        int mulResult = (fact[i] * n) + carry;
        fact[i] = mulResult % 10;
        carry = mulResult / 10;
    }

    while (carry != 0) {
        fact.push_back(carry % 10);
        carry /= 10;
    }
}

string factorial(int n) {
    vector<int> fact(1, 1);

    for (int i = 2; i <= n; i++)
        multiply(fact, i);

    string result = "";
    for (int i = fact.size() - 1; i >= 0; i--) {
        result += to_string(fact[i]);
    }

    return result;
}


int main() {
    int n;
    cin >> n;
    if (n < 1) {
        cout << "Invalid input" << '\n';
        return 0;
    }
```

```
    cout << factorial(n) << '\n';
    return 0;
}
```

P3-Justify the Time Complexity of give pseudo Code with proper reasoning;

a-)

```
int j = 0;
for(int i = 0; i < n; ++i) {
while(j < n && arr[i] < arr[j]) {
j++;
}
}
```

b-)

```
int a = 0, i = N;
while (i > 0) a += i;
i /= 2;
}
```

a) O(n)

- If predicate condition of while loop is false then i will increment, otherwise j will increment. In either case, the outer for loop will terminate after at most 2n steps.

b) O(log n)

- After every iteration of the while loop, i become half of its previous value.

P4-What is the additional benefit of a circular queue? Illustrate with at least 2 practical examples (pro-
gramming/application) where a circular queue is used. Write the enque and deque functions/pseudocode

(not the program) for array and linked list based implementation of a circular queue.

Benefits of Circular queue :

1. **Good utilization of memory :**

Example :

- Consider a simple Queue and a Circular Queue of size 5.
- Simple Queue = {1, 2, 3, 4, 5}. front is at index = 0 and rear is at index = 4
- Circular Queue = {1, 2, 3, 4, 5} front is at index = 0 and rear is at index = 4

- Now If we remove the first element in the queue which is 1(pointed by the front of both queues) then the state of both of the queues will be as follows.

- Simple Queue = {-, 2, 3, 4, 5}. front is at index = 1 and rear is at index = 4
- Circular Queue = {-, 2, 3, 4, 5} front is at index = 1 and rear is at index = 4

- Now if we want to insert 6 in both the queues then their state will be as follows.

- Not possible to insert in the simple queue as the rear has reached at the end.
- Circular Queue = {6, 2, 3, 4, 5} front is at index = 1 and rear is at index = 0. This is because the rear will move circularly from index 4 to index 0. Thus vacant positions can be utilized which was not possible in linear queue.

2. **Elements can be inserted again :**

Example :

- Consider a simple Queue and a Circular Queue of size 5.
- Simple Queue = {1, 2, 3, 4, 5}. front is at index = 0 and rear is at index = 4
- Circular Queue = {1, 2, 3, 4, 5} front is at index = 0 and rear is at index = 4

- Now If we remove the first 3 elements in the both queues then the state of both of the queues will be as follows.

- Simple Queue = {-, -, -, 4, 5}. front is at index = 3 and rear is at index = 4
- Circular Queue = {-, -, -, 4, 5} front is at index = 3 and rear is at index = 4

- Now if we want to insert 6, 7 in both the queues then their state will be as follows.

- Not possible to insert in the simple queue as the rear has reached at the end. So, in a linear queue, once an element is removed then it is not possible to insert new elements at those indices.
- Circular Queue = {6, 7, -, 4, 5} front is at index = 3 and rear is at index = 1. This is because the rear will move in a circular manner from index 4 to index 1 and thus the vacant positions can be filled again unlike in a linear queue.

Pseudocode :

1. **Array Based Implementation :**

Enqueue :
Enqueue(arr, front, rear, capacity, num):

```
If (rear + 1) % capacity == front :
        return
If rear == -1:
        front = 0
        rear = 0
        arr[rear] = num
        return
rear  = (rear + 1) % capacity
arr[rear] = num
```

Dequeue :
Dequeue(arr, front, rear, capacity):

```
If rear == -1:
        return
If front == rear :
        num = arr[front]
        front = -1
        rear = -1
num = arr[front]
front  = (front + 1) % capacity
return num
```

2. **Linked List Based Implementation :**

Enqueue :

```
Enqueue(front, rear, num, size, capacity):

If size == 0:
        front = rear
        rear.data = num
        size++
If capacity == size:
        return
rear = rear.next
rear.data = num
size++
```

Dequeue :

```
Dequeue(front, rear, size, capacity):

If size == 0:
        return
If size == 1:
        num = front.data
        front = rear
        size = 0
        return num
num = front.data
front = front.next
size--
return num
```

References:

1. https://www.mvorganizing.org/what-is-the-reason-for-using-a-circular-queue-instead-of-a-regular-one