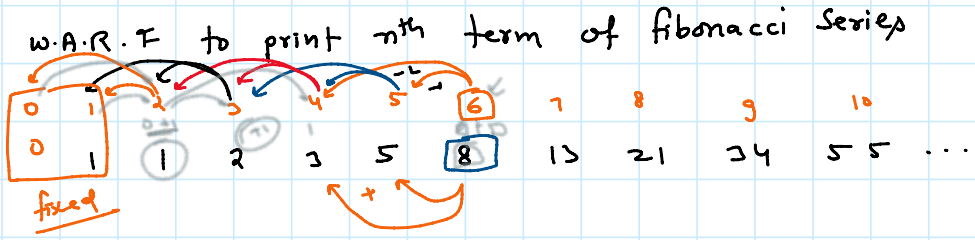
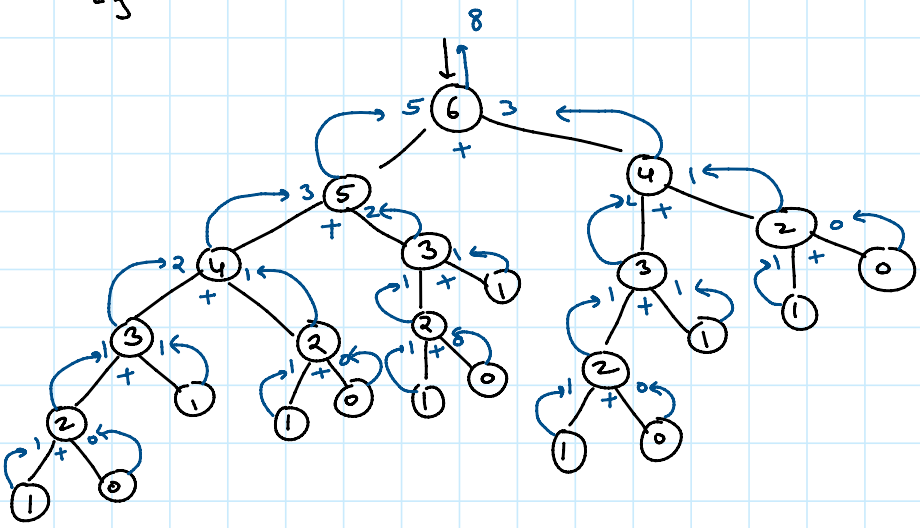
DAY 13

08 May 2024 01:01 PM




```
int fibo(int n)
{
    if(n==0 || n==1)
        return n;

    return fibo(n-1) + fibo(n-2);
}
```

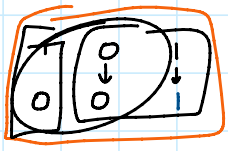


.

```cpp
#include<iostream>
using namespace std;
int fact(int n)
{
    if(n==1 or n==0)
        return n;
    return fact(n-1) + fact(n-2);
}
int main()
{
    cout<<fact(10);
    return 0;
}
```

Tribonacci Series :-

Tribonacci Series :-



2	3	4	5	6	7	8	...	n
1	2	4	7	13	24	44	...	

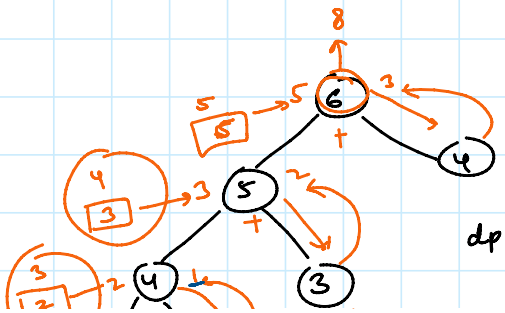
W.A.R.F to print nth term of tribonacci series –

```
int tribo(int n)
{
    if(n==0 || n==1)
        return n;
    if(n==-1)
        return 0;

    if (n==-1 || n==0)
        return 0;
    if(n==1)
        return 1;

    return tribo(n-1) + tribo(n-2) + tribo(n-3);
}
```

```cpp
#include<iostream>
using namespace std;
int tribo(int n)
{
    if(n==1 or n==0)
        return n;
    if(n==-1)
        return 0;
    return tribo(n-1) + tribo(n-2) + tribo(n-3);
}
int main()
{
    cout<<tribo(10);
    return 0;
}
```

Dynamic Programming (DP)

fibonacci Series nth term :-

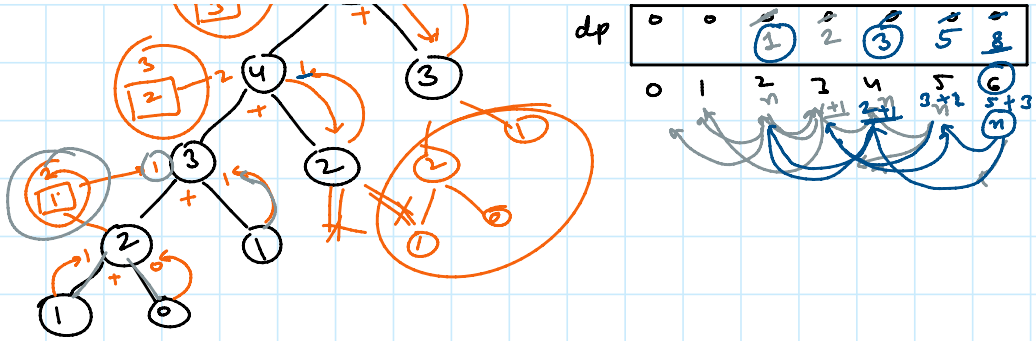


(11)

size 7

dp

0	0	1	2	3	5	8
0	1	2	3	4	5	6



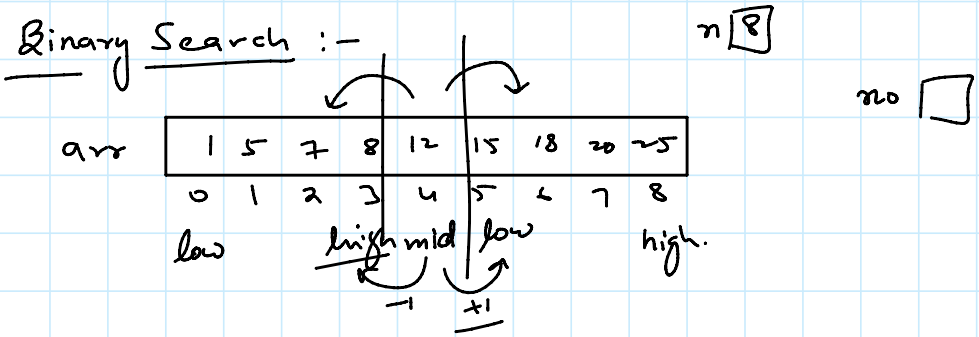

```
#include<iostream>
using namespace std;
int count=0;
int fact(int n,int dp[])
{
    count++;
    if(n==1 or n==0)
        return n;
    if(dp[n]==0)
        dp[n] = fact(n-1,dp) + fact(n-2,dp);
    return dp[n];
}
int main()
{
    int n;
    cout<<"Enter no of a term:";
    cin>>n;
    int dp[n+1]={0};
    cout<<fact(n,dp)<<endl;
    cout<<count;
    return 0;
}
```

Apply DP on tribonacci series nth term

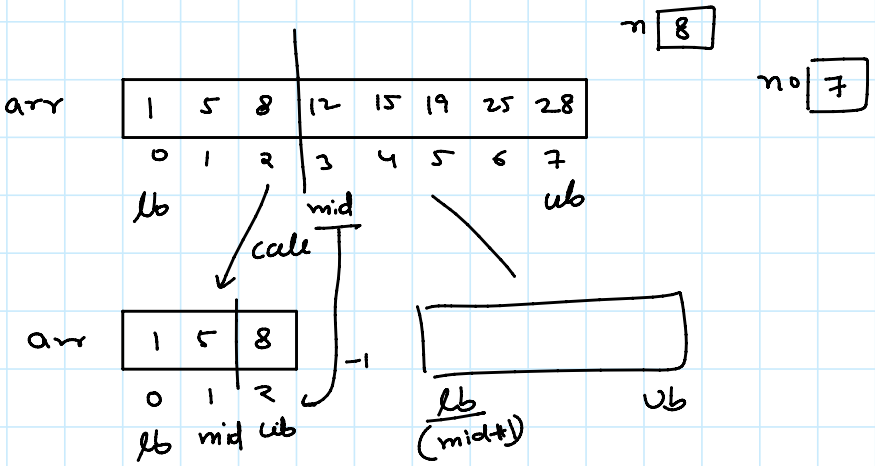
```
#include<iostream>
using namespace std;
int tribo(int n,int dp[])
{
    if(n==1 or n==0)
        return n;
    if(n==-1)
        return 0;
    if(dp[n]==0)
        dp[n] = tribo(n-1,dp) + tribo(n-2,dp) + tribo(n-3,dp);
    return dp[n];
}
int main()
{
    int n;
    cout<<"Enter no of a term:";
```

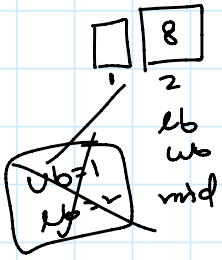
```
    cin>>n;
    int dp[n+1]={0};
    cout<<tribo(n,dp)<<endl;
    return 0;
}
```

Binary Search :-



Binary Searching using recursion






```
bool BinarySearch (int arr[], int lb, int ub, int no)
{
    if (lb>ub)
        return false;
    int mid = (lb+ub)/2;
```

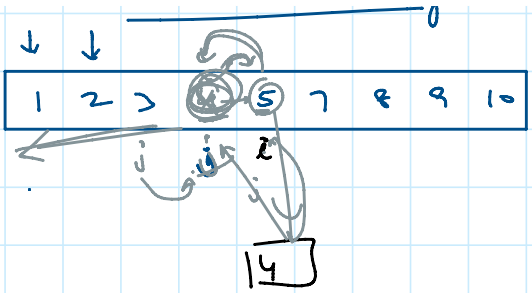
```
        return false;
    int mid = (lb+ub)/2;
    if(arr[mid]==no)
        return true;
    if(no < a[mid])
        return BinarySearch(arr, lb, mid-1, no);
    else
        return BinarySearch(arr, mid+1, ub, no);
}
```

.

```
#include<iostream>
using namespace std;
bool binarySearch(int arr[], int lb, int ub,int no)
{
    if(lb>ub)
        return false;
    int mid=(lb+ub)/2;
    if(arr[mid]==no)
        return true;
    if(no < arr[mid])
        return binarySearch(arr,lb,mid-1,no);
    else
        return binarySearch(arr,mid+1,ub,no);
}
int main()
{
    int arr[]={1,5,9,10,15,17,19,22,25};
    int n=sizeof(arr)/sizeof(int);
    if(binarySearch(arr,0,n-1,190))
        cout<<"Found";
    else
        cout<<"Not found";
    return 0;
}
```

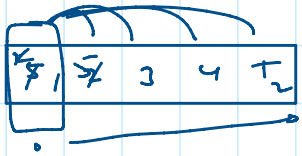
Sorting :-

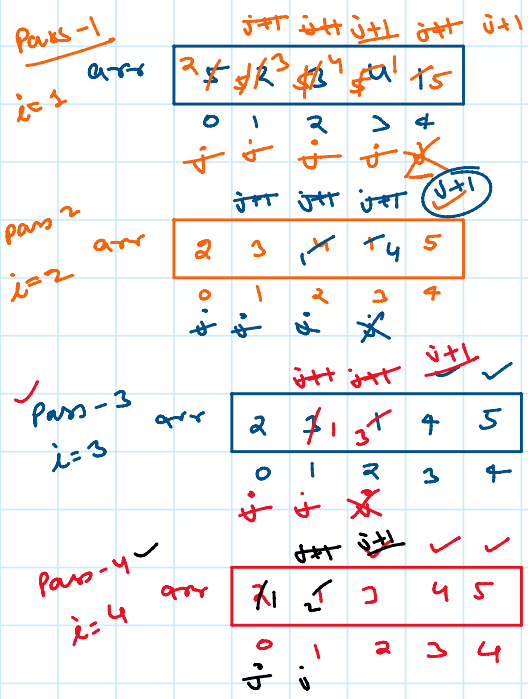
Insertion Sorting :-




```
for(i=1 to <n)
{
    no = arr[i];
    for(j=i-1 ; j>=0 && arr[j]>no ; j--)
    {
        arr[j+1] = arr[j];
    }
    arr[j+1] = no;
}
```

Bubble Sorting :-



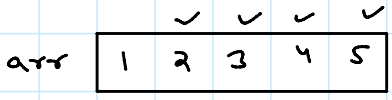



```
if(arr[j] > arr[j+1])
    swap(arr[j], arr[j+1])
```

i	j
1	0-3
→ 2	0-2
3	0-1
4	0-0

i = 1 to < n ++

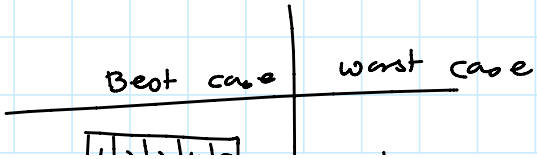
j = 0 to < n-i ++



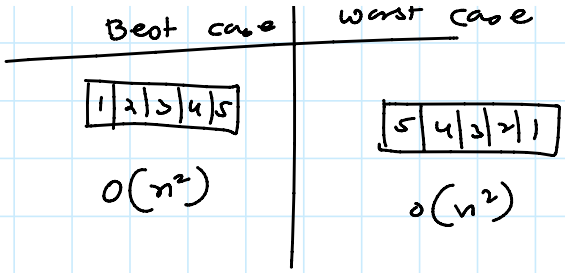
↑ Sorted Array

I. method

```
#include<iostream>
using namespace std;
void BubbleSort(int arr[], int n)
{
    for(int i=1;i<n;i++)
    {
```

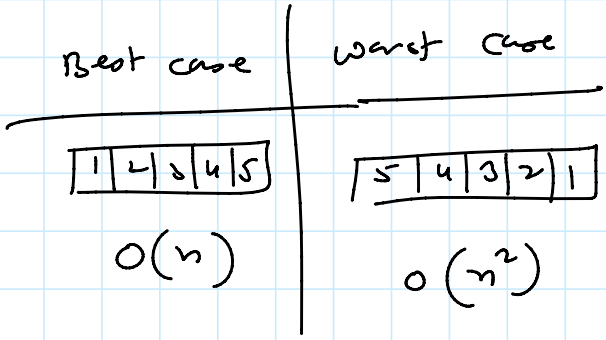


```
{
    for(int i=1;i<n;i++)
    {
        for(int j=0;j<n-i;j++)
        {
            if(arr[j]>arr[j+1])
                swap(arr[j],arr[j+1]);
        }
    }
}
int main()
{
    int arr[]={5,1,3,2,4};
    int n=sizeof(arr)/sizeof(int);
    BubbleSort(arr,n);
    for(int i:arr)
        cout<<i<<" ";
    return 0;
}
```

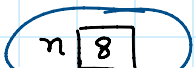


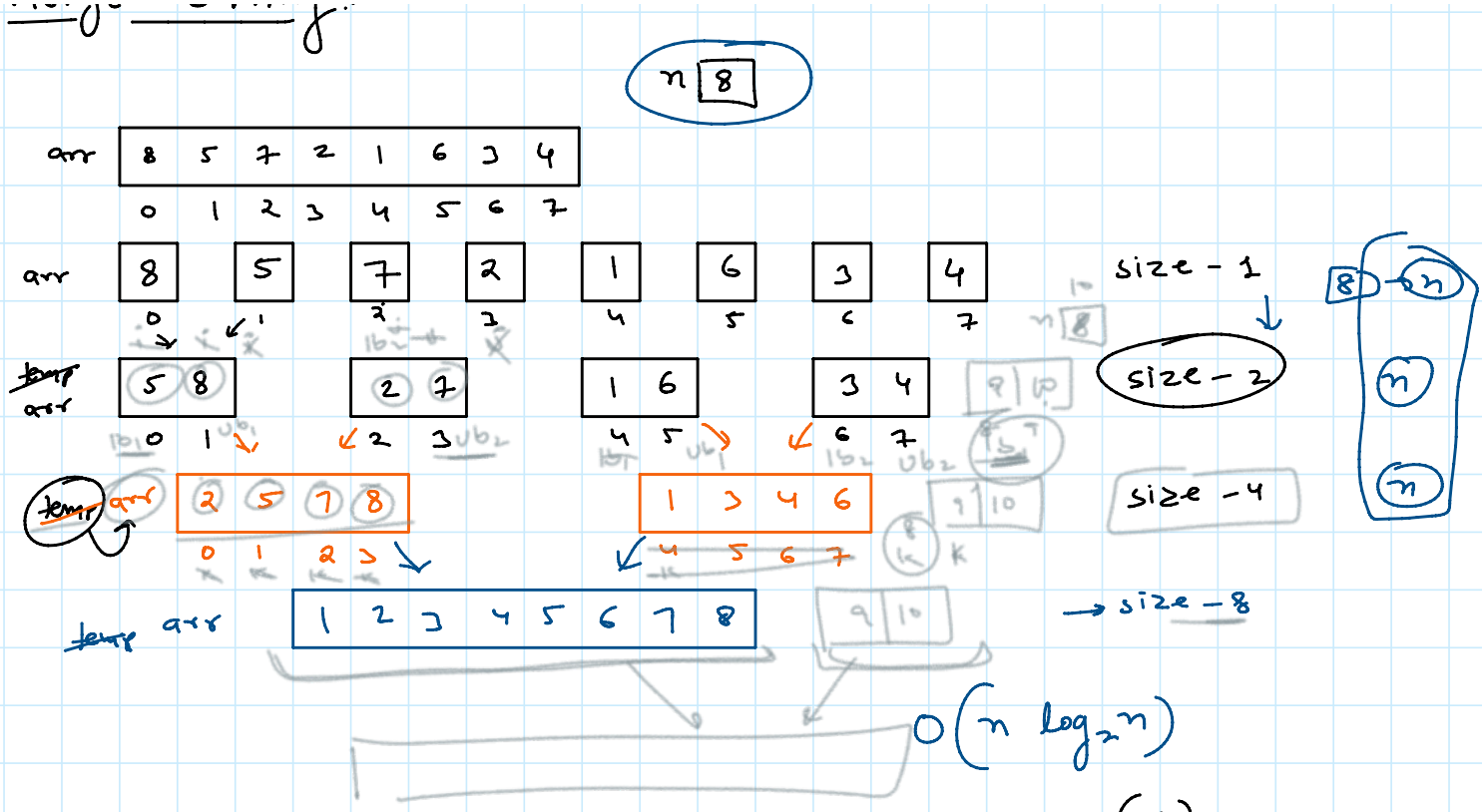
II method

```
#include<iostream>
using namespace std;
void BubbleSort(int arr[], int n)
{
    bool flag=true;
    for(int i=1;i<n and flag;i++)
    {
        flag=false;
        for(int j=0;j<n-i;j++)
        {
            if(arr[j]>arr[j+1])
            {
                swap(arr[j],arr[j+1]);
                flag=true;
            }
        }
    }
}
int main()
{
    int arr[]={5,1,3,2,4};
    int n=sizeof(arr)/sizeof(int);
    BubbleSort(arr,n);
    for(int i:arr)
        cout<<i<<" ";
    return 0;
}
```



Merge Sorting :-

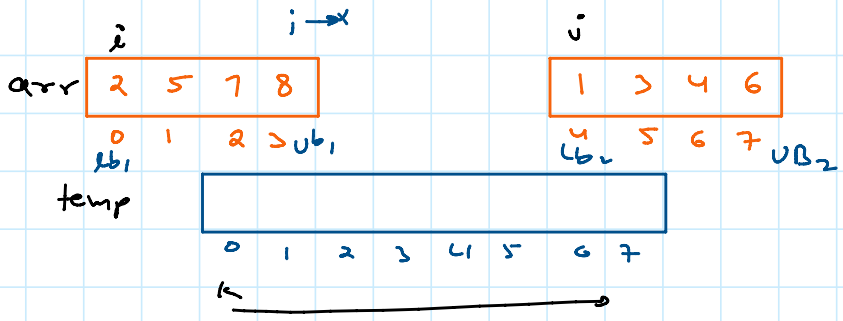




① Copy from temp to arr

```
for(i=0; i<n; i++)
    arr[i] = temp[i];
```

② merge to parts :-



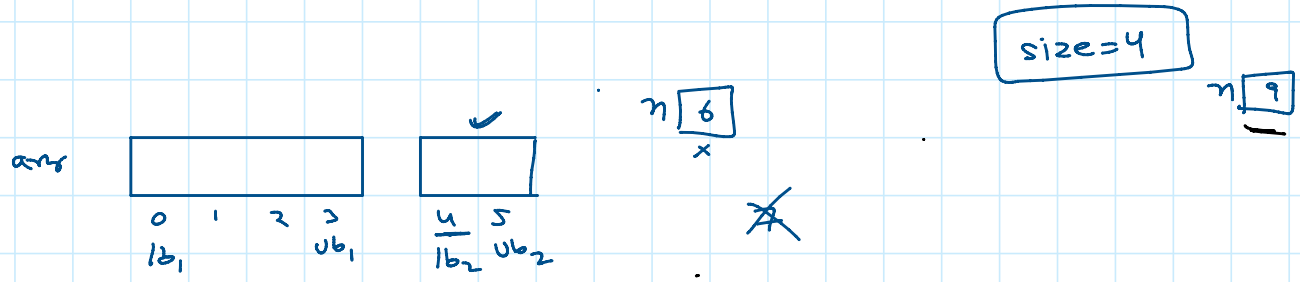

```
for(i=lb1, j=lb2; i<=ub1 and j<=ub2; k++)
{
    if(arr[i] < arr[j])
    {
        temp[k] = arr[i];
        i++;
    }
    else
    {
        temp[k] = arr[j];
        j++;
```

```
    {
        temp[k] = arr[j];
        j++;
    }
```

```
while (i<=ub1)
{
    temp[k++] = arr[i++];
}
```

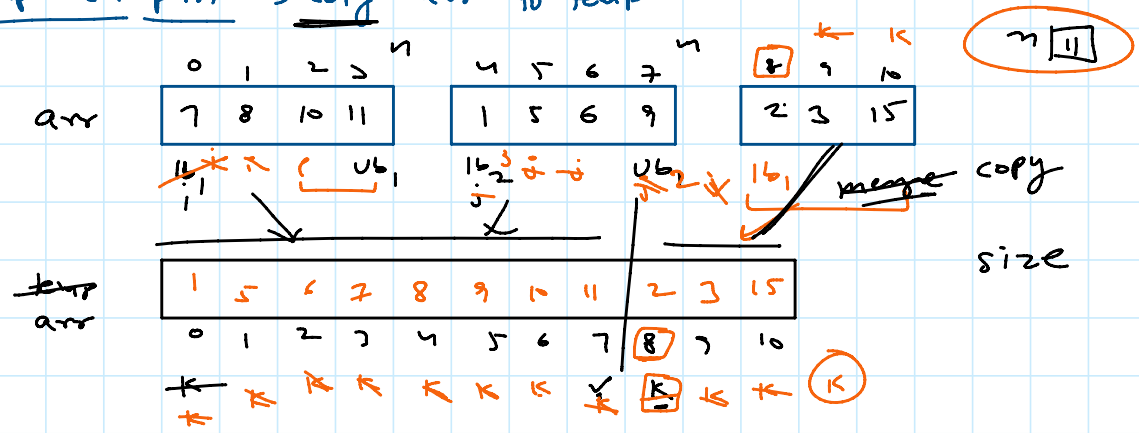
```
while (j<=ub2)
{
    temp[k++] = arr[j++];
}
```

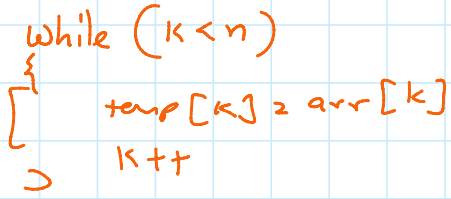
③ Partition :-



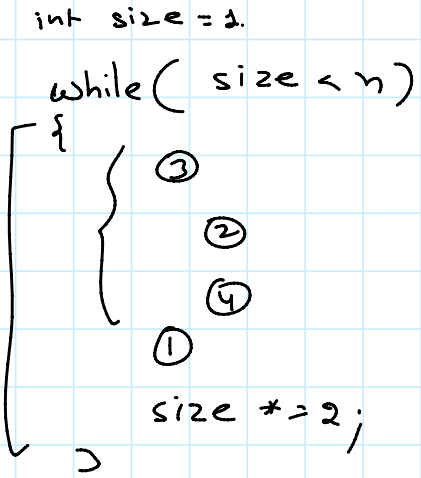

```
k = 0;
lb1 = 0;
while( (lb1+size) < n)
{
    lb2 = lb1+size;
    ub1 = lb2-1;
    ub2 = (ub1+size)<n ? ub1+size : n-1;
    // marge to parts
    lb1 = ub2+1
}
```

④ unpaired part → copy arr to temp





⑤ size



.

```
#include<iostream>
using namespace std;
void mergeSort(int arr[], int n)
{
    int size=1,i,j,k,lb1,ub1,lb2,ub2;
    int temp[n];
    while(size<n)
    {
        k=0;
        lb1=0;
        while((lb1+size)<n)
        {
            lb2=lb1+size;
            ub1=lb2-1;
            ub2=(ub1+size)<n?ub1+size:n-1;
            for(i=lb1,j=lb2;i<=ub1 and j<=ub2;k++)
            {
                if(arr[i]<arr[j])
                    temp[k]=arr[i++];
                else
                    temp[k]=arr[j++];
            }
            while(i<=ub1)
                temp[k++]=arr[i++];
            while(j<=ub2)
                temp[k++]=arr[j++];
            lb1=ub2+1;
        }
        while(k<n)
```

```
            {
                temp[k]=arr[k];
                k++;
            }
            for(i=0;i<n;i++)
                arr[i]=temp[i];
            size *= 2;
        }
    }
    int main()
    {
        int arr[]={5,1,3,2,4,8,7,6,9};
        int n=sizeof(arr)/sizeof(int);
        mergeSort(arr,n);
        for(int i:arr)
            cout<<i<<" ";
        return 0;
    }
```