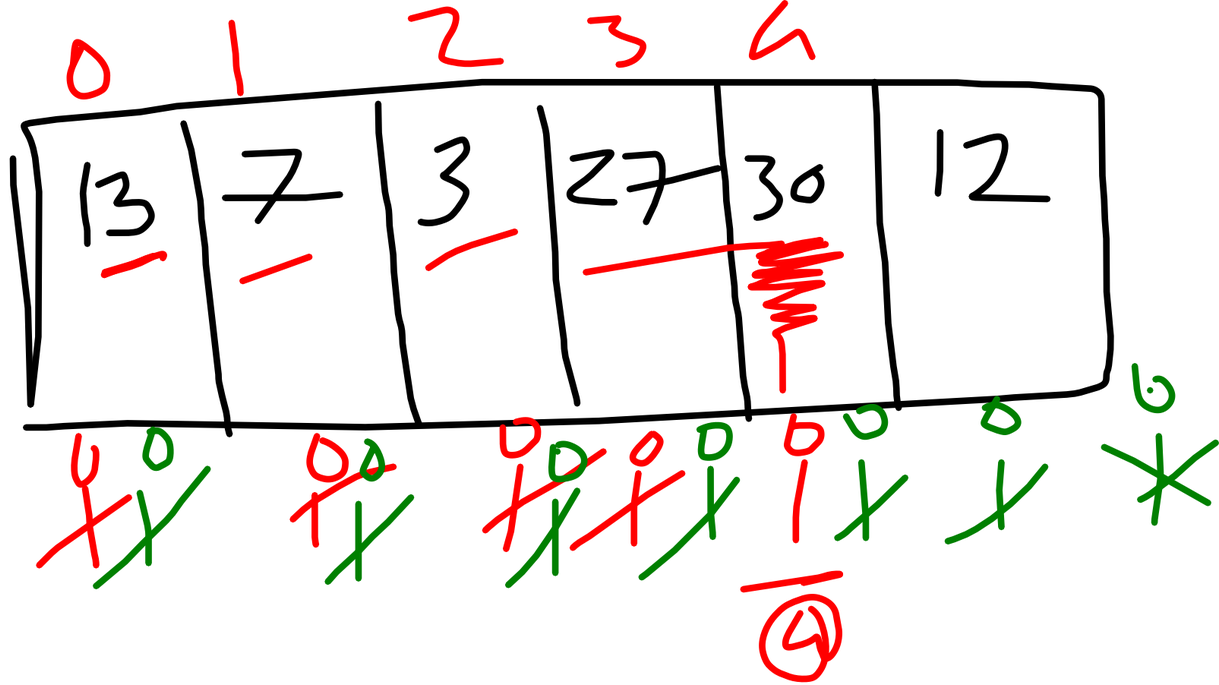
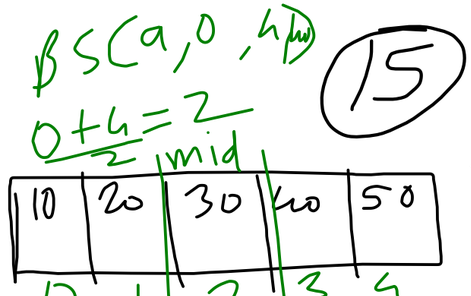
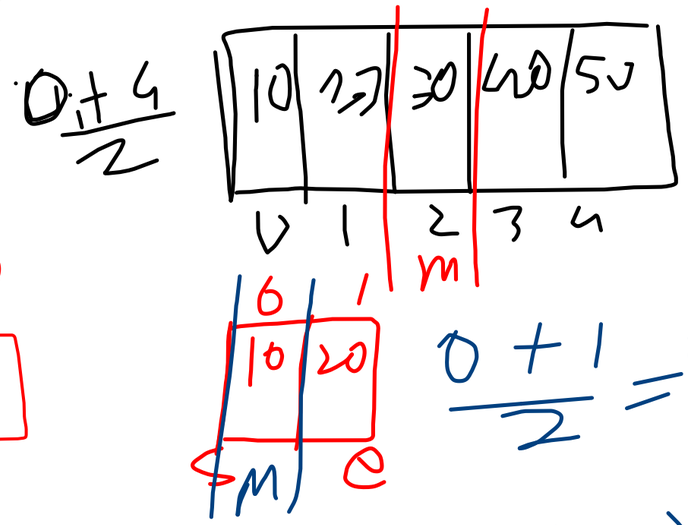
```
int sequential_search(int a[],int key)
    {
        for(int i=0;i<a.length;i++)
            {
                if(key==a[i])
                    return i;//found
            }
        return -1;//not found
    }
```



Key=46



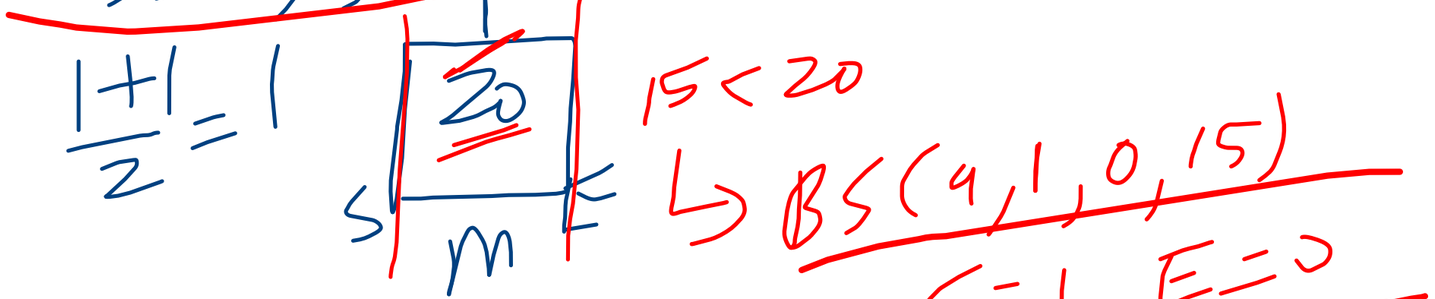


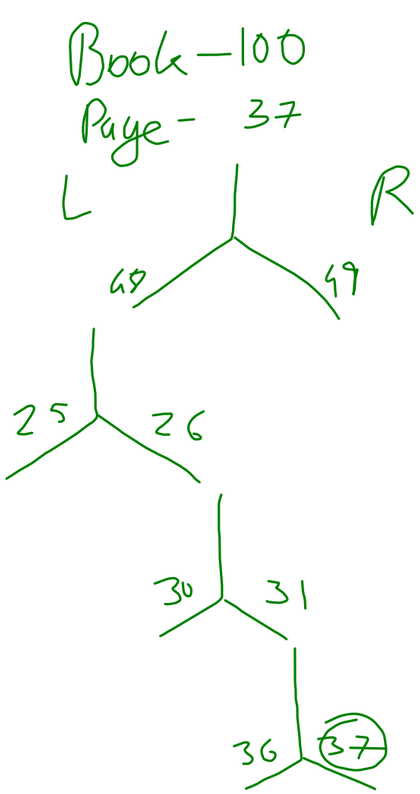

```
int binary_search(int a[],int start,int end,int key)
{
  if(start<=end)
    {  int mid=(start+end)/2;
       if(a[mid]==key)
          return mid;
       else
          {  if(key<a[mid)
                return binary_search(a,start,mid-1,key);
             else
                return binary_search(a,mid+1,end,key);
          }
    }
  else
    {return (-1);
    }
}
```

key = 40

0

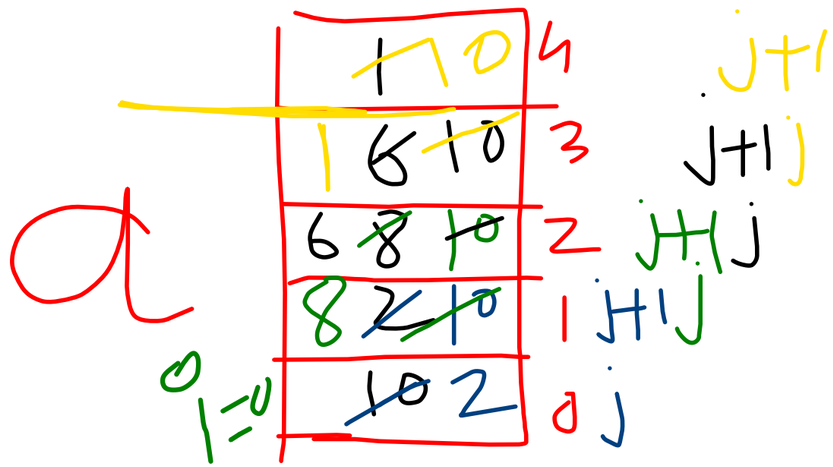
4



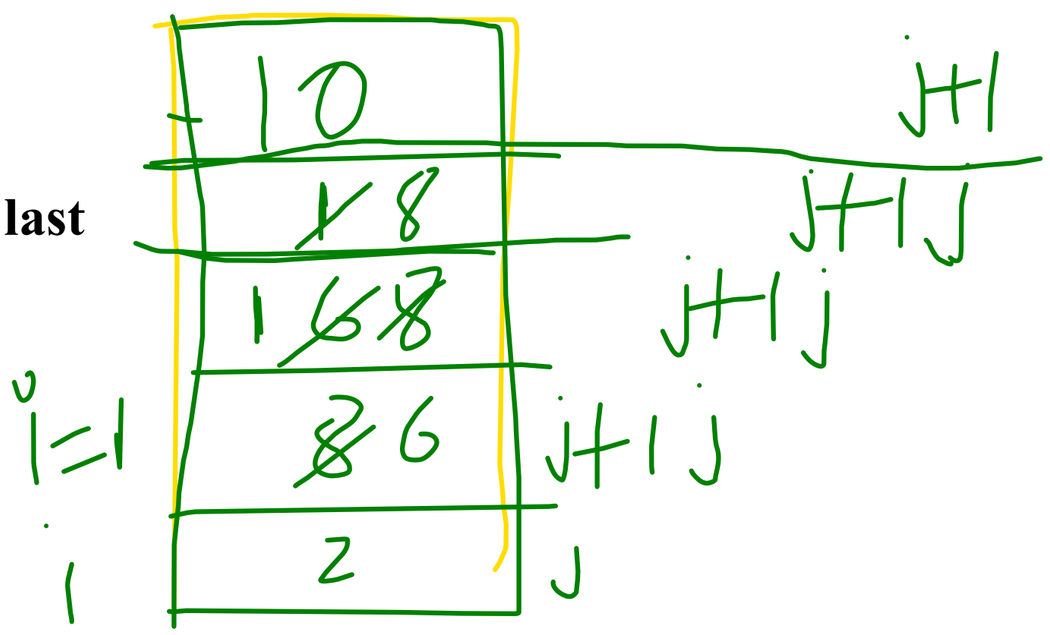


sorting:Data in some specific order.
Always first step for faster searching.
Technique: remember why it is called, what is compared, and its time complexity.

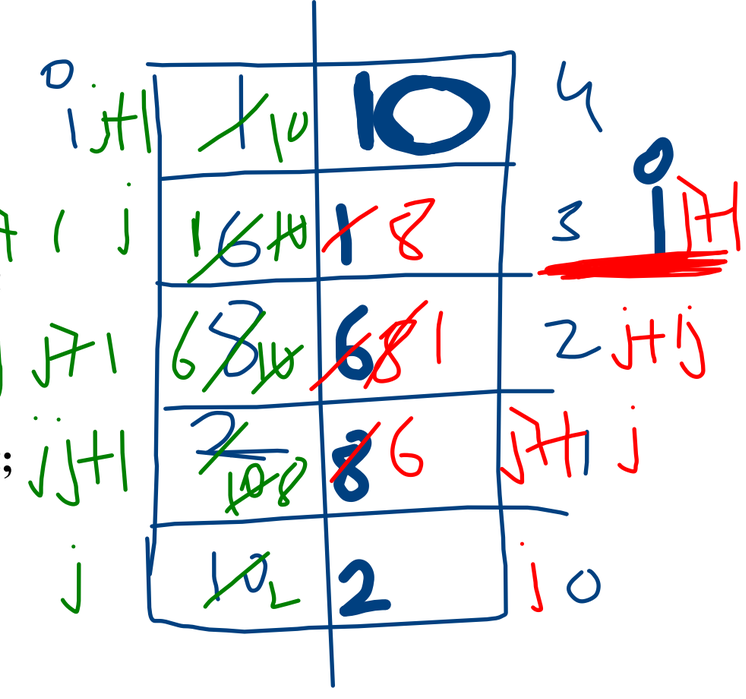
Bubble sort:



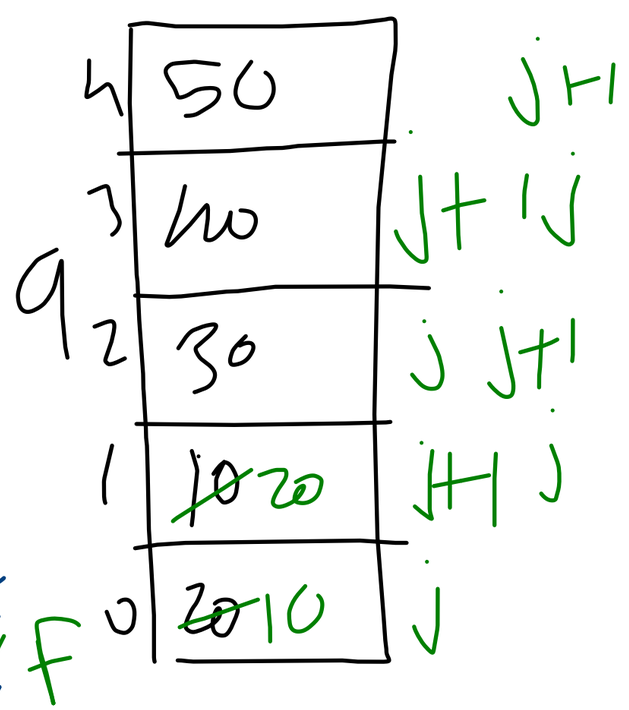
```
void bubble_sort(int a[])
    {
        for(int i=0;i<a.length-1;i++)//5-1=4 0,1,2,3
            {
                for(int j=0;j<a.length-1;j++)//j stops at second last
                    {
                        if(a[j]>a[j+1])
                            {
                              int t=a[j];a[j]=a[j+1];a[j+1]=t;
                            }
                    }
            }
```



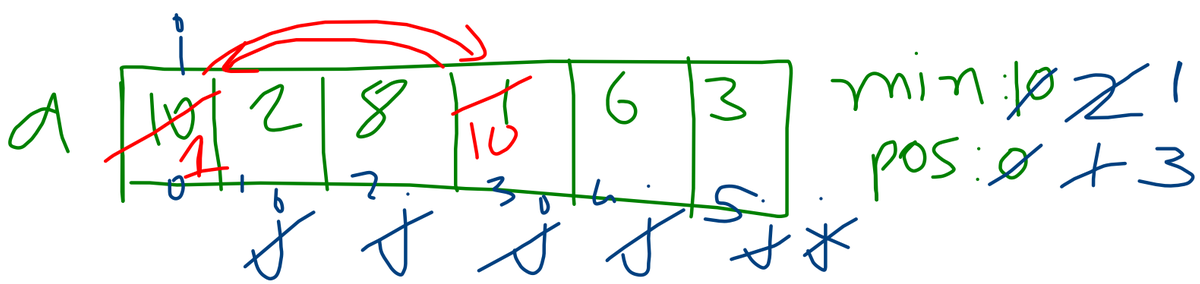
```
void optimized_bubble_sort(int a[])
    {
        for(int i=a.length-1;i>0;i--)//limit passes
            {
                for(int j=0;j<i;j++)//j stops at second last
                    {
                        if(a[j]>a[j+1])
                            {
                             int t=a[j];a[j]=a[j+1];a[j+1]=t;
                            }
                    }
            }
```



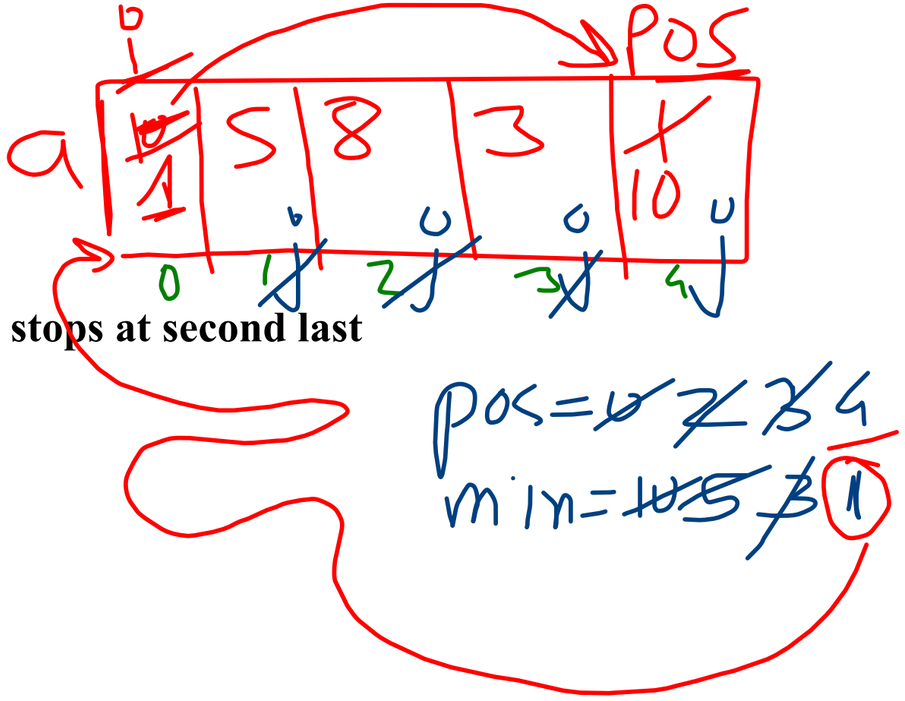
```
void optimized_bubble_sort(int a[])
    {   boolean done=ture;
        for(int i=a.length-1;i>0;i--)//limit passes
            {
                done=true;
                for(int j=0;j<i;j++)//j stops at second last
                    {
                        if(a[j]>a[j+1])
                            {
                             int t=a[j];a[j]=a[j+1];a[j+1]=t;
                             done=false;
                            }
                    }
                if(done==ture)
                    break;
            }
```

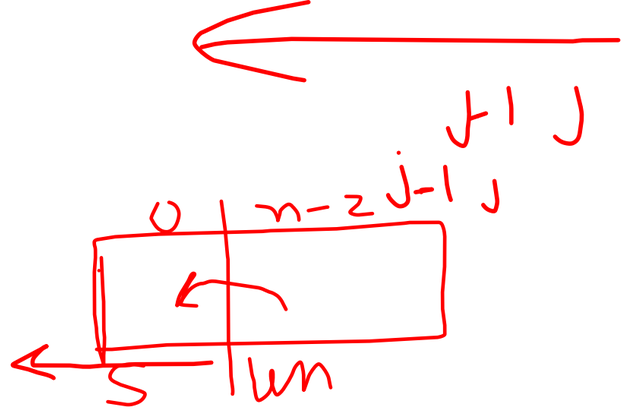


Selection sort:
in this we select minimum element in each pass and place it at ith right location.
in pass 1 1 smallest
in pass 2 2nd smallest

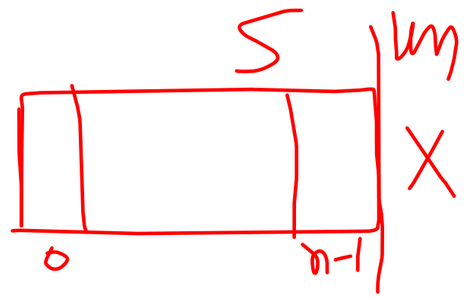



```
void selection_sort(int a[])
    {
        int min,pos;
        for(int i=0;i<a.length-1;i++)
            {
                min=a[i];pos=i;//reference
                for(int j=i+1;j<a.length;j++)//j stops at second last
                    {
                        if(a[j]<min)
                            {//update
                              min=a[j];pos=j;
                            }
                    }
            //swap
            a[pos]=a[i];a[i]=min;
            }
```



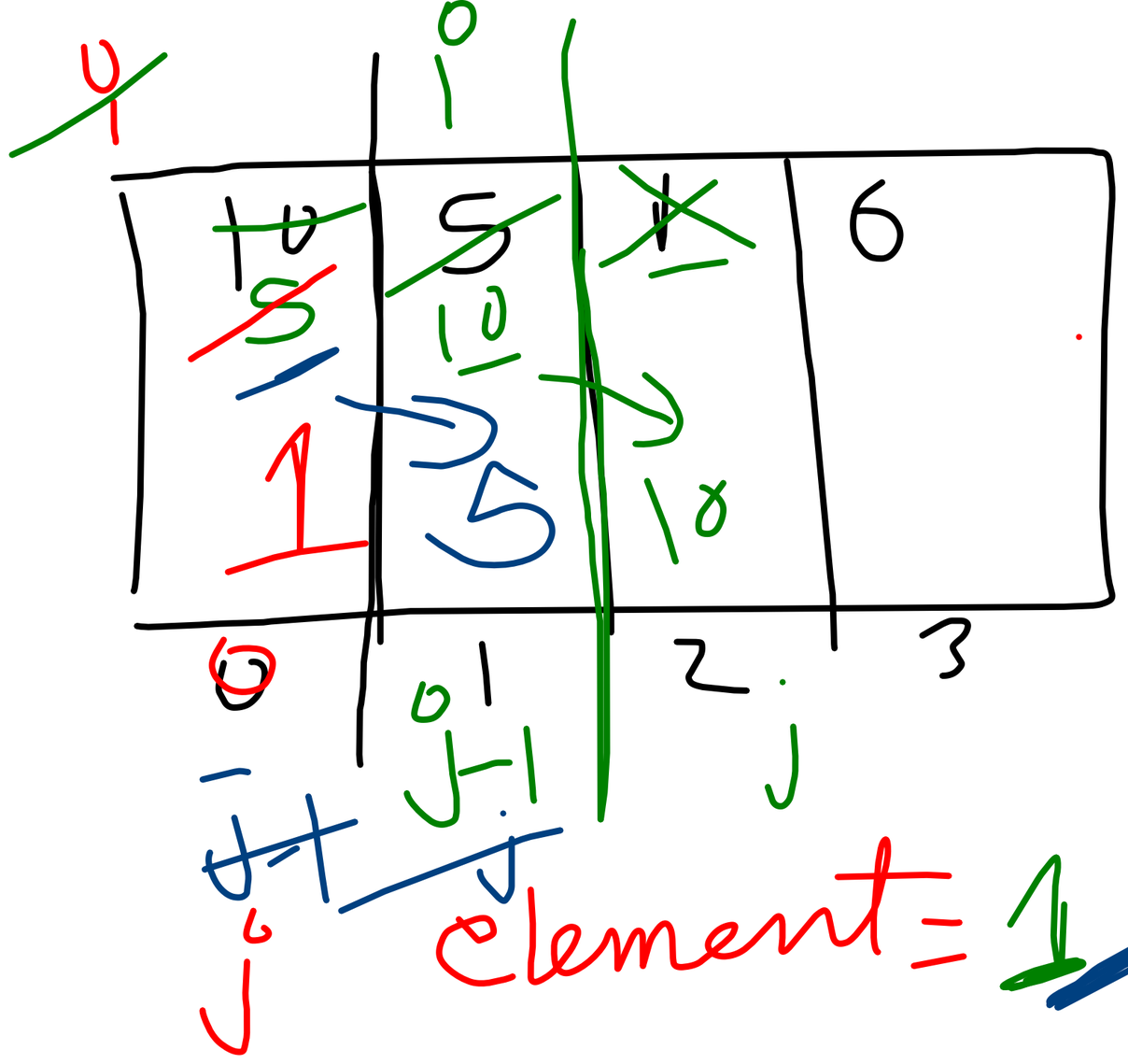


Insertion sort:
in this array is div in 2 parts:
1:sorted
2:unsorted
in each pass one element from unsorted is INSERTED in sorted part at right location



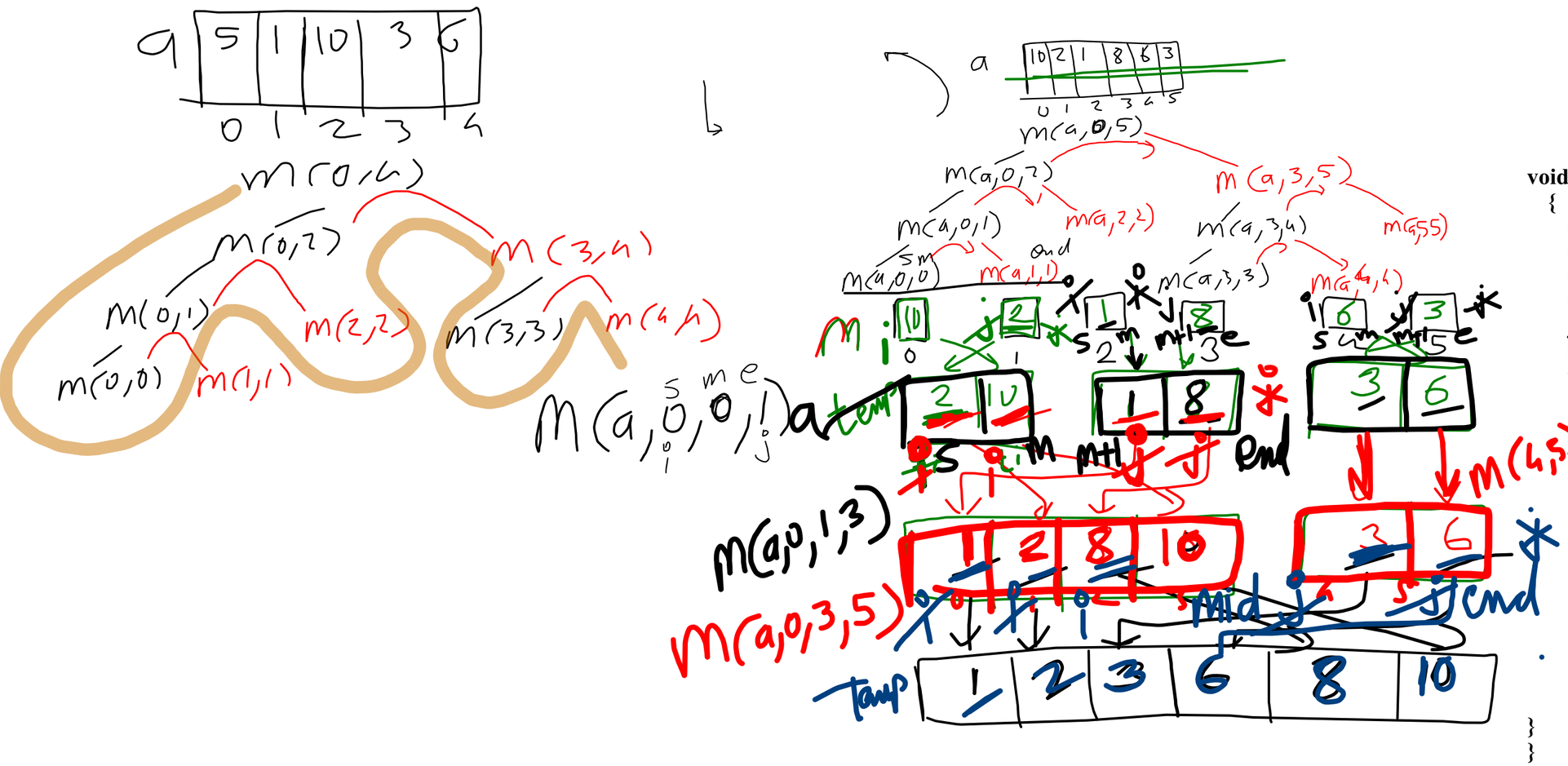

```
void insertion_sort(int a[])
    {
        int element;
        for(int i=0;i<a.length-1;i++)
            {
                element=a[i+1];
                while(j>0 && a[j-1]>element)
                    {   //move back
                        a[j]=a[j-1];
                        j--;//move ahead
                    }
            //insert
            a[j]=element;
            }
}
```

```
void insertion_sort(int a[])
    {
        int element;
        for(int i=0;i<a.length-1;i++)
            {
                element=a[i+1];
                j=i+1;
                while(j>0 && a[j-1]>element)
                    {   //move back
                        a[j]=a[j-1];
                        j--;//move ahead
                    }
            //insert
            a[j]=element;
            }
}
```



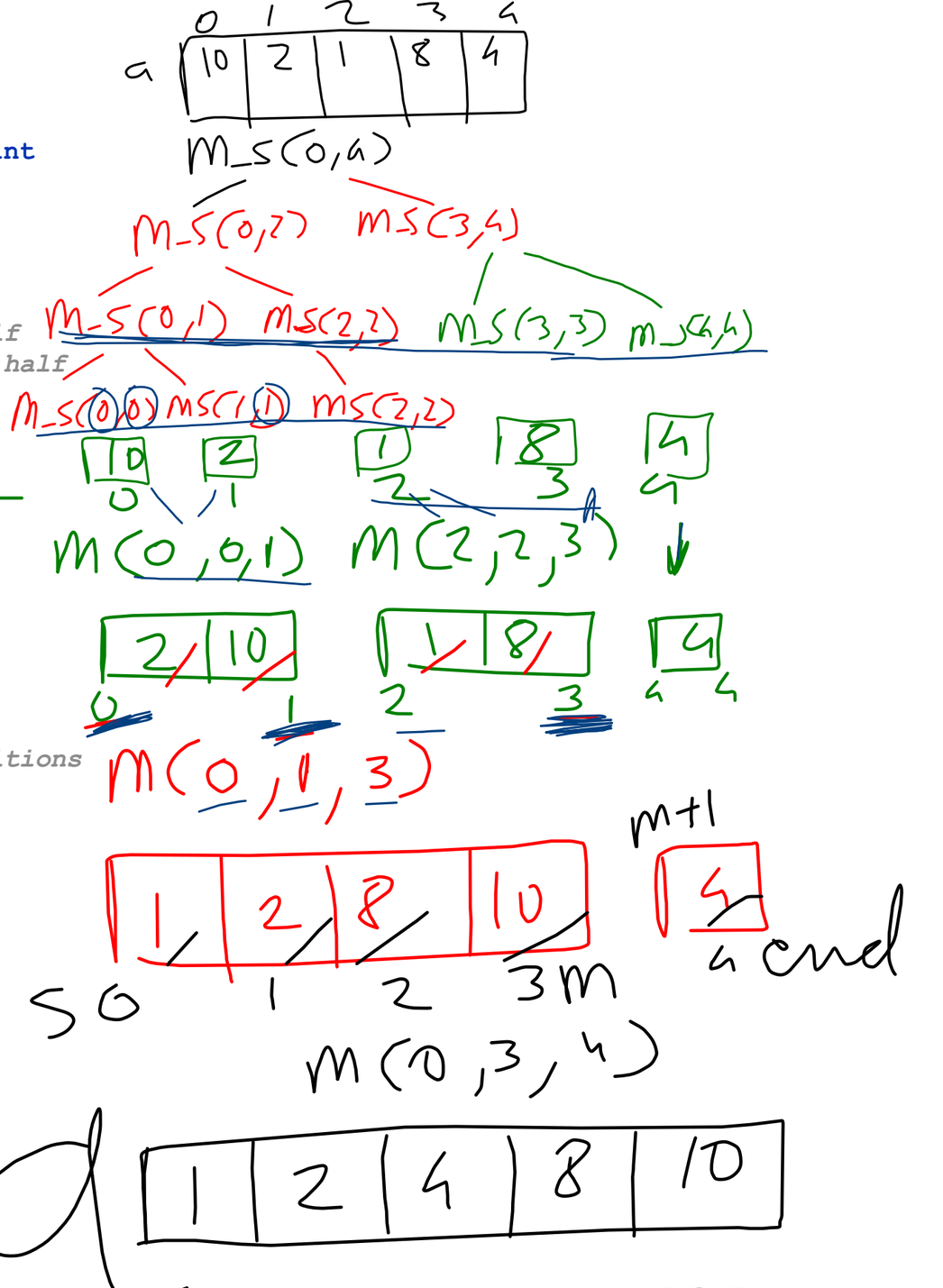
merge sort:
Famous for being divide and conquer.

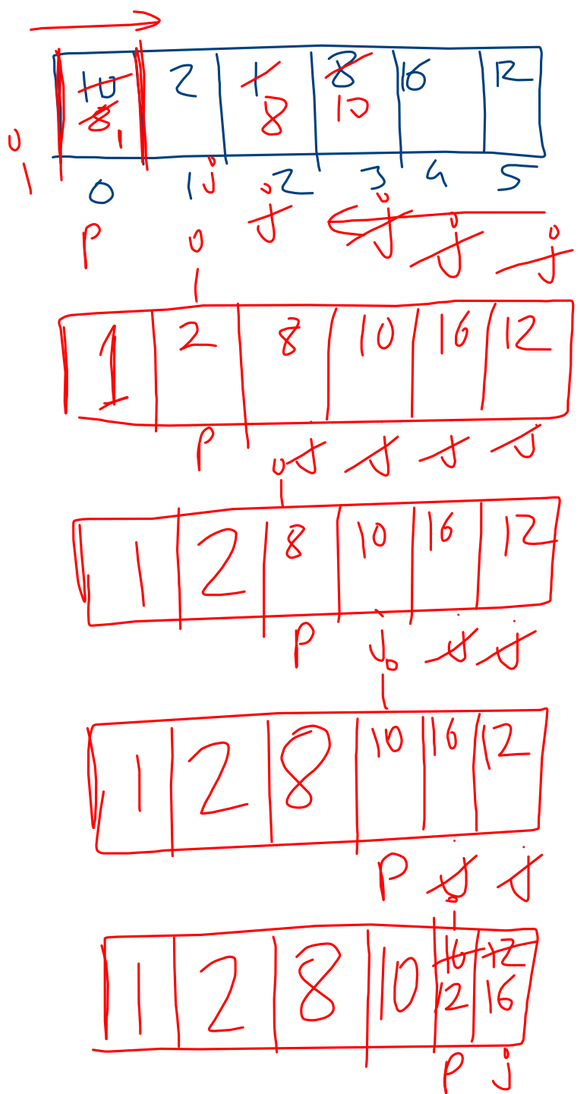
We start with a single array of N elements.
Divide them again and again from the mid,
and we stop when we reach N arrays of size one.



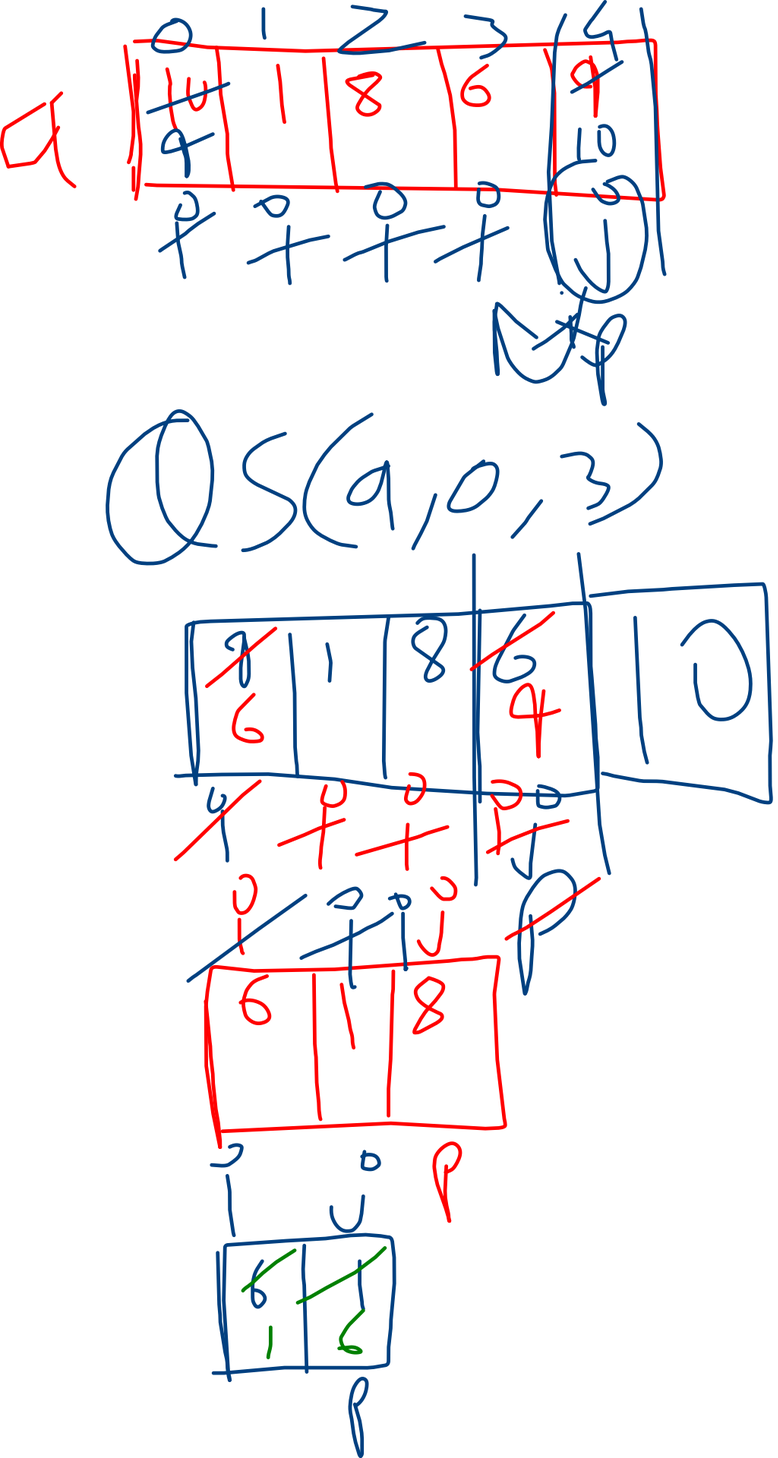

```
void merger(int a[],int start,int mid,int end)
  {
     int i,j;//index for block
     int tindex,t[];//temp array
     i=start;
     tindex=start;
     j=mid+1;
     t=new int[a.length];
     while(i<=mid && j<=end)//boundry conditions
        {
          if(a[i]<a[j])
             temp[tindex++]=a[i++];
          else
             temp[tindex++]=a[j++];
     }
     while(i<=mid)
          temp[tindex++]=a[i++];
     while(j<=end)
          temp[tindex++]=a[j++];
     //copy temp to a again for round i+1
     for(i=start;i<=end;i++)
        a[i]=temp[i];
  }
}
```

```java
static void merge_sort(int a[],int start,int
end)
{
    if(start<end)//size-1 stop
    {
        int mid=(start+end)/2;
        merge_sort(a,start,mid);//left half
        merge_sort(a,mid+1,end);//  right half
        merger(a,start,mid,end);
    }
}
static void merger(int a[],int start,int
mid,int end)
{
    int i,j;//index for block
    int tindex,temp[];//temp array
    i=start;
    tindex=start;
    j=mid+1;
    temp=new int[a.length];
    while(i<=mid && j<=end)//boundry conditions
    {
        if(a[i]<a[j])
            temp[tindex++]=a[i++];
        else
            temp[tindex++]=a[j++];
    }
    while(i<=mid)
        temp[tindex++]=a[i++];
    while(j<=end)
        temp[tindex++]=a[j++];
    //copy temp to a again for round i+1
    for(i=start;i<=end;i++)
        a[i]=temp[i];
}
```




10 2 4 8 16 R
8 1 8 10
0 1 2 3 4 5
p i j j j
1 2 8 10 16 12
p i j j j
1 2 8 10 16 12
p j
1 2 8 10 16 12
p j
1 2 8 10 12 16
p j

```
void quick_sort(int a[],int start,int end)
{
    int i,j,pivot,temp;
    i=start;
    pivot=start;
    j=end;
    while(i<j)
    {
        while(a[i]<a[pivot])
            i++;
        while(a[j]>a[pivot])
            j--;
        if(i<j)
            {
                temp=a[i];a[i]=a[j];a[j]=temp;
             }
     }
    //if pivot in start
    if(i<=end)
        quick_sort(a,i+1,end);
    //if pivot in end
    if(j>start)
        quick_sort(a,start,j-1);
}
```



```
void quick_sort(int a[],int start,int end)
{
    int i,j,pivot,temp;
    i=start;
    pivot=start;
    j=end;
    while(i<j)
    {
        while(a[i]<a[pivot])
            i++;
        while(a[j]>a[pivot])
            j--;
        if(i<j)
            {
                temp=a[i];a[i]=a[j];a[j]=temp;
            }
    }
    //if pivot in start
    if(i<=end)
        quick_sort(a,i+1,end);
    //if pivot in end
    if(j>start)
        quick_sort(a,start,j-1);
}
```