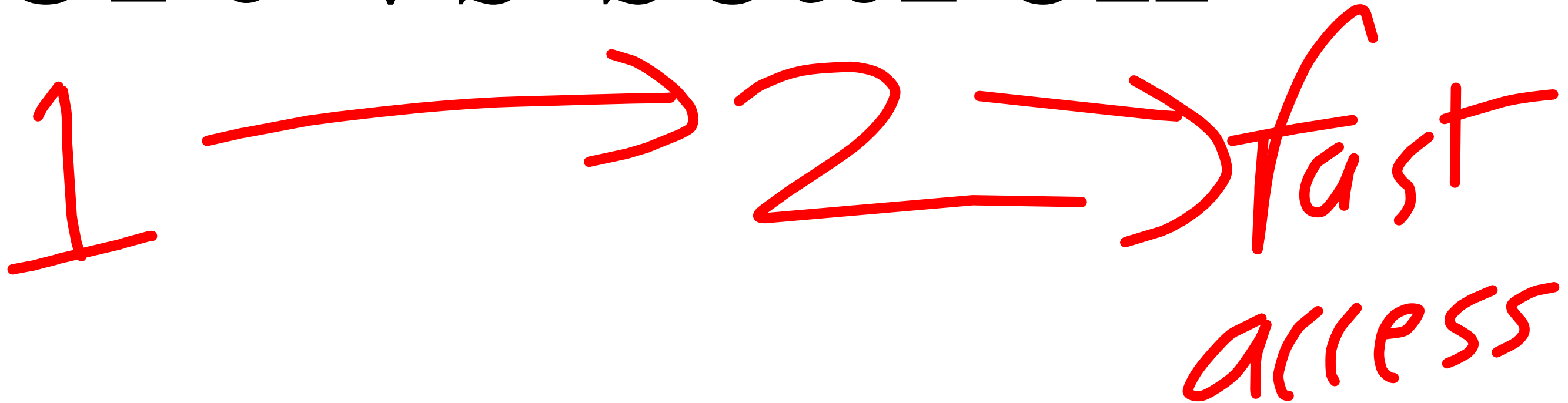


external(data on secondary storage) and internal(ram)

sort vs search



Bubble sort: basic/slowest logic:

neighbours are compared with each other and if needed will swap. $a[j]$ compared with $a[j+1]$ if needed swap.
in each pass right element is bubbled on top(length-i)

$n-1 \leftarrow \text{sort } n$
passes



```
for(i=0;i<a.length-1;i++) //given n-1 passes
```

```
{
```

```
for(j=0;j<a.length-1;j++) //compare and swap if needed
```

```
{
```

```
if(a[j]>a[j+1])
```

```
{
```

```
temp=a[j];
```

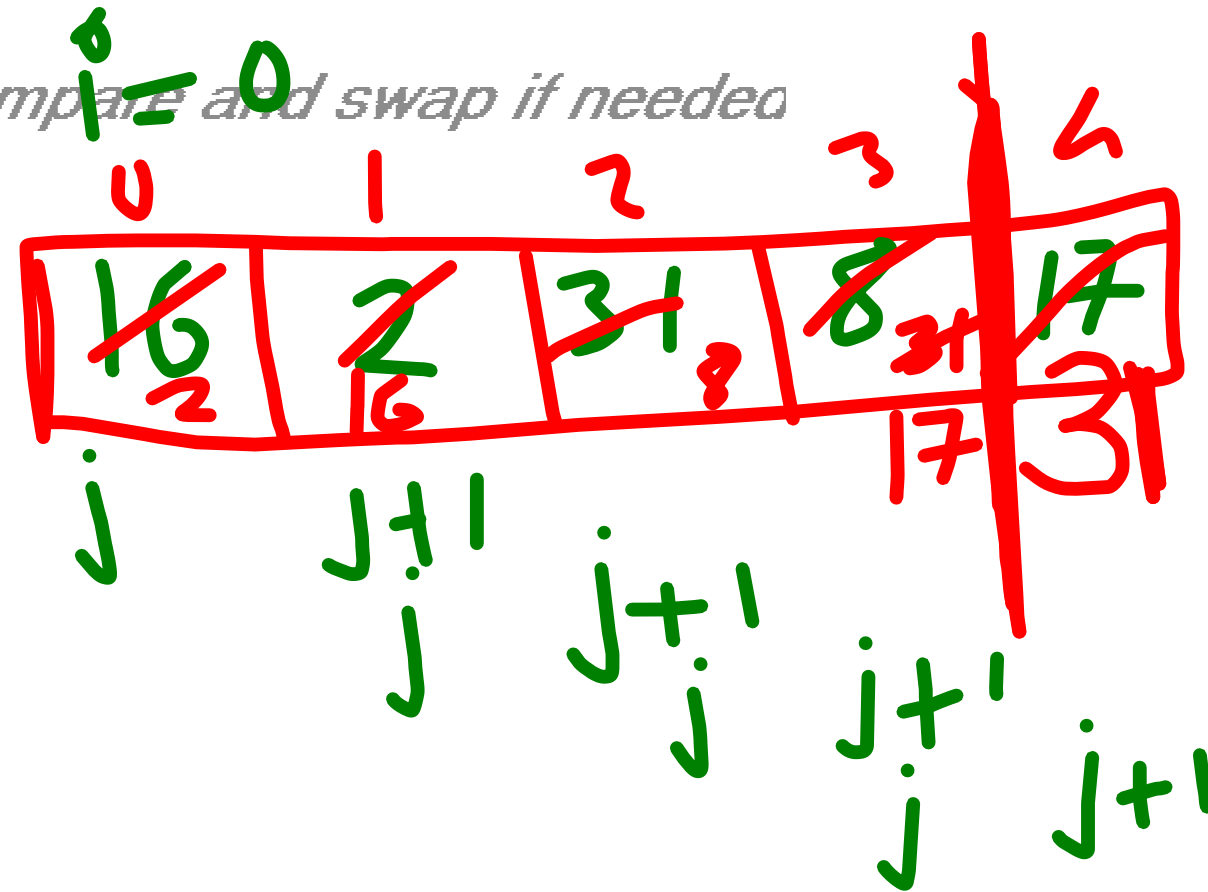
```
a[j]=a[j+1];
```

```
a[j+1]=temp;
```

```
}
```

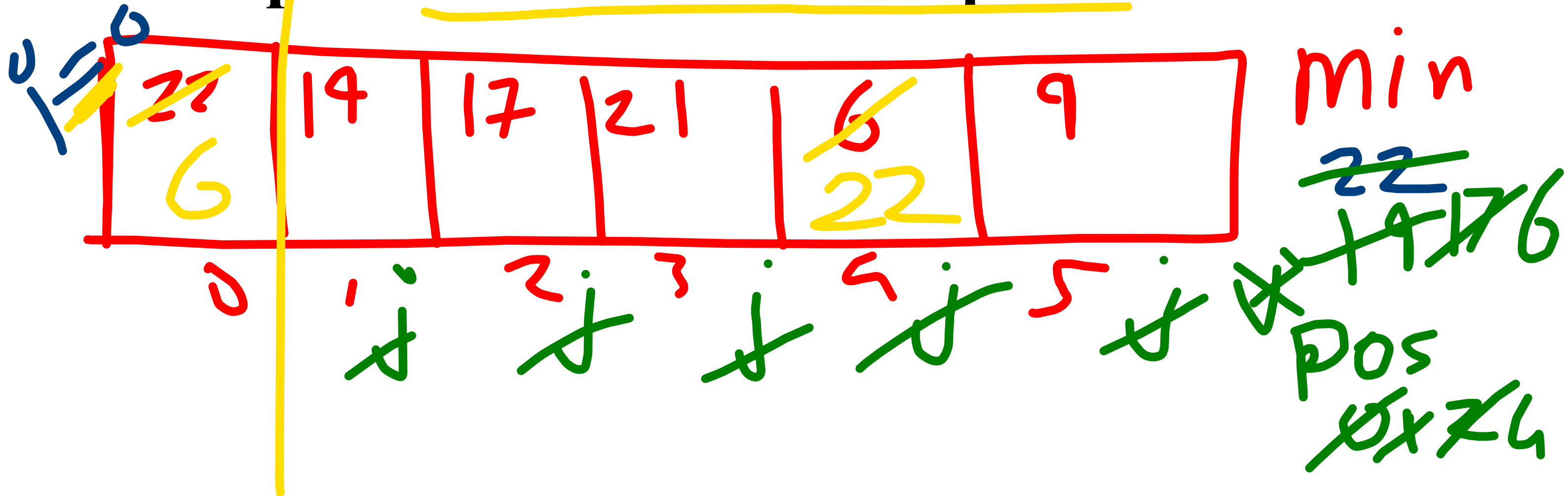
```
}
```

```
}
```



Selection sort:

**in each pass smallest element is selected and
then kept at ith location in ith pass.**



```
static void selection_sort(int a[])
```

```
{
```

```
    int i,j,min,pos;
```

```
    for(i=0;i<a.length-1;i++) //given n-1 passes
```

```
    {
```

```
        min=a[i]; //ref
```

```
        pos=i; //ref
```

```
        for(j=i+1;j<a.length;j++)
```

```
        {
```

```
            if(a[j]<min)           
```

```
            {
```

```
                pos=j;
```

```
                min=a[j];
```

```
            }
```

```
        }
```

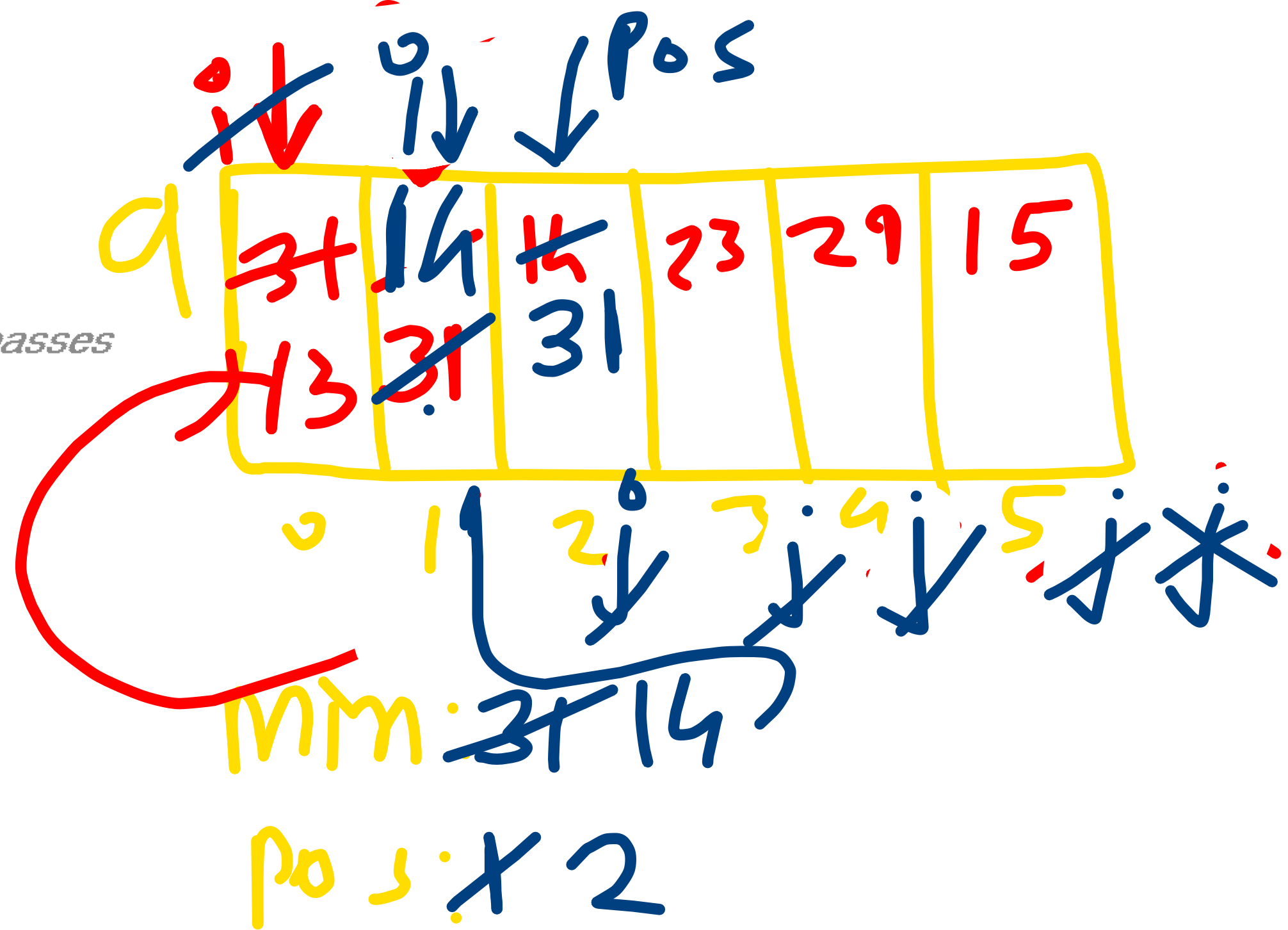
```
        //swap
```

```
        a[pos]=a[i];
```

```
        a[i]=min;
```

```
    }
```

```
}
```



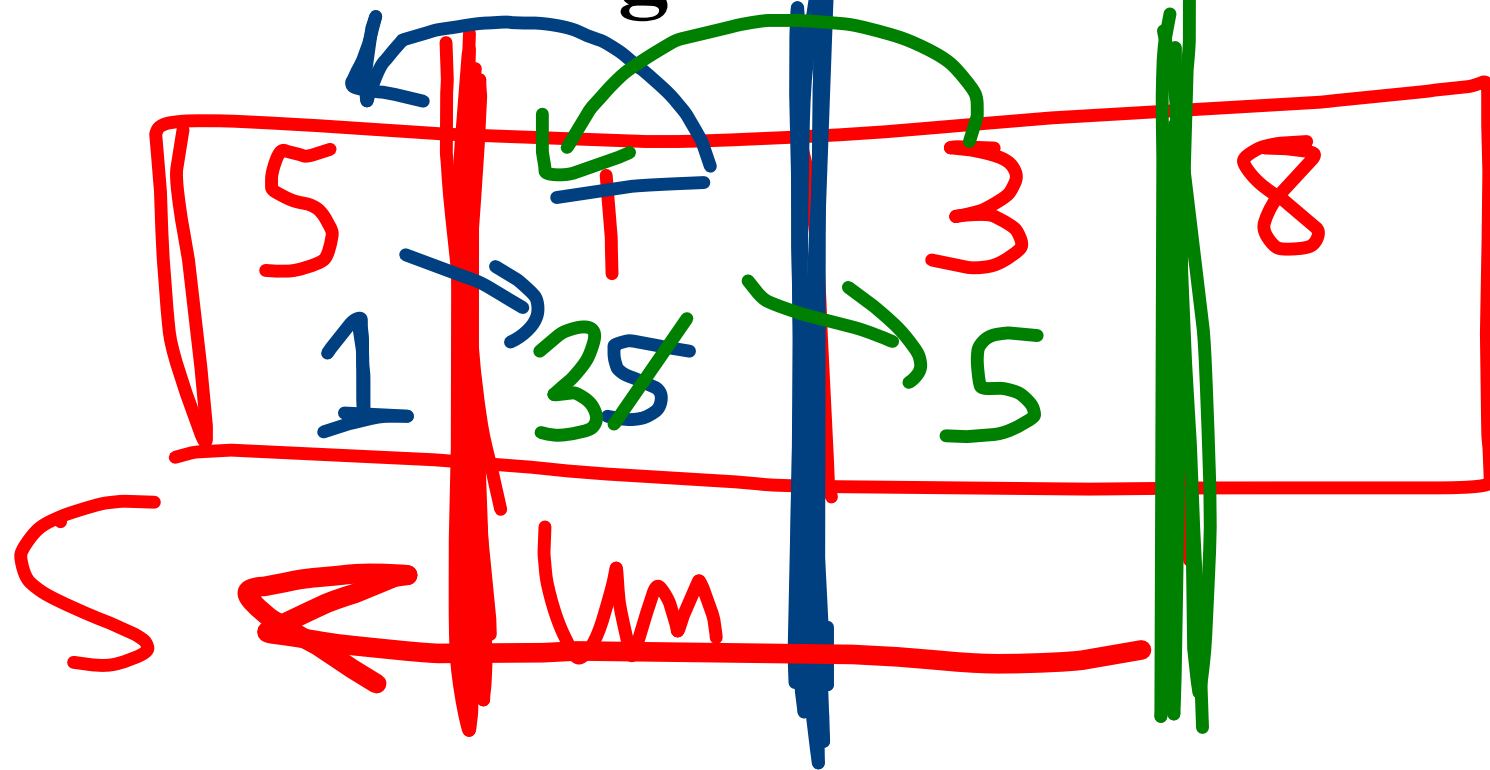
insertion sort:

array is broken in 2 parts

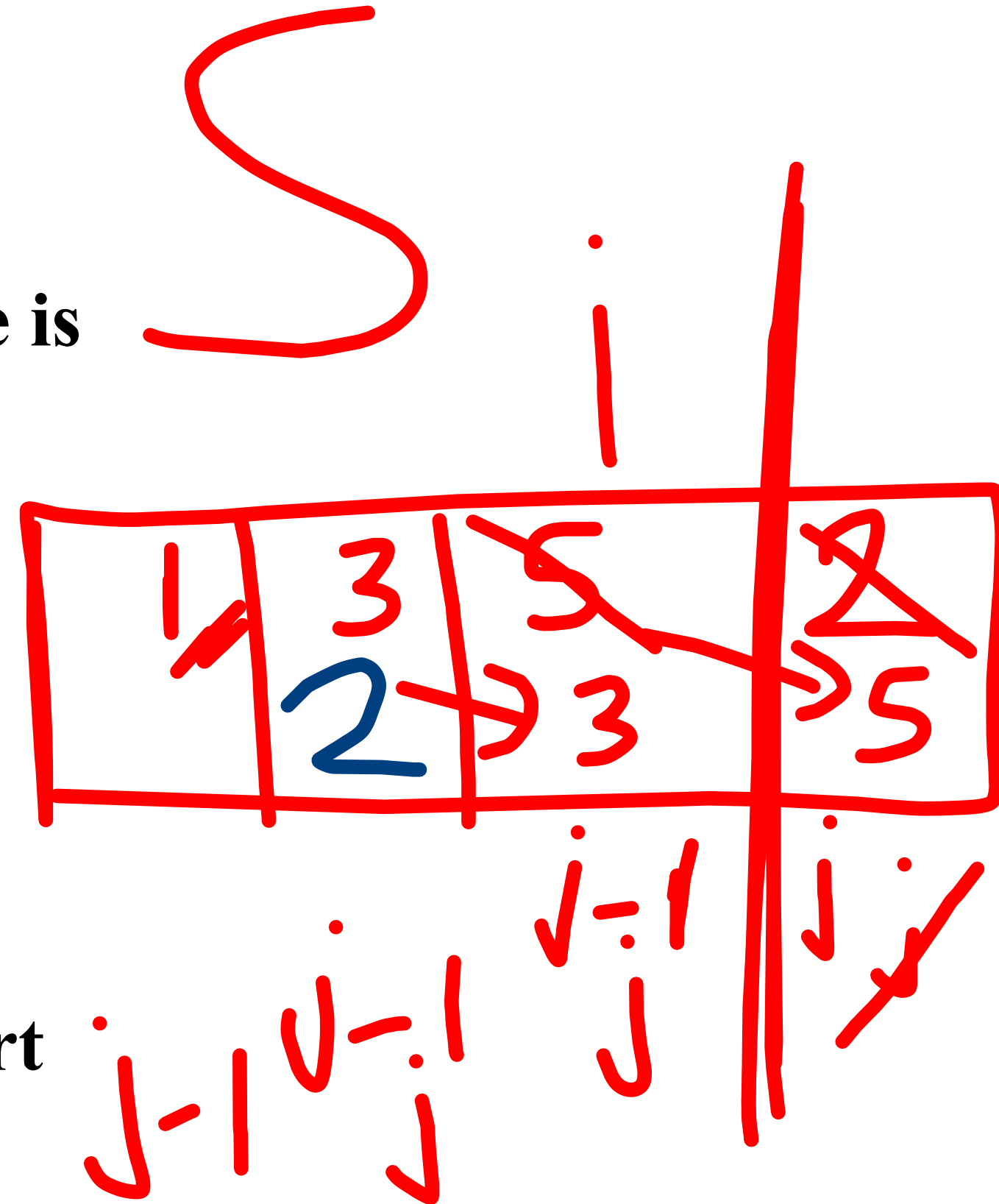
1 sorted

2 unsorted

in each pass an element from unsorted side is inserted at right location in sorted part.



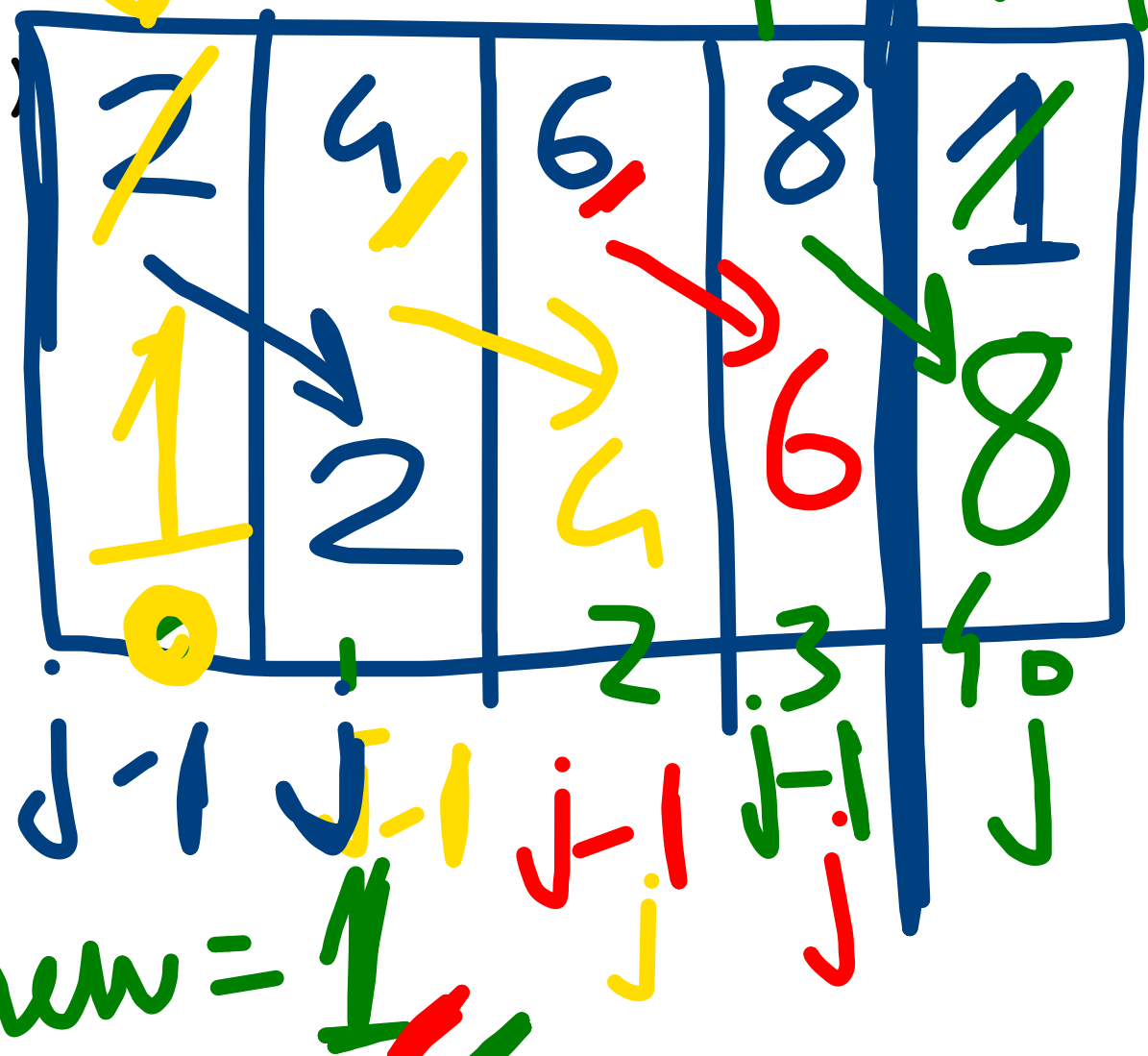
will stop when all element are in sorted part



```

static void insertion_sort(int a[])
{
    int i,j,new_element;
    for(i=0;i<a.length-1;i++) //
given n-1 passes
    {
        new_element=a[i+1]; //ref
        j=i+1; //unsorted starts
from
        while(j>0 &&
new_element<a[j-1])
        {
            a[j]=a[j-1]; //pullback
            j--;
        }
        a[j]=new_element;
    }
}

```



Quick sort

among the fastest sort.

sort from both sides using ref of pivot

pivot is ref either 1st,last,mid

it takes ref of pivot and rearranges everything with ref to pivot

-sorting is done by keeping all elements $<$ pivot before it and all element \geq pivot after it



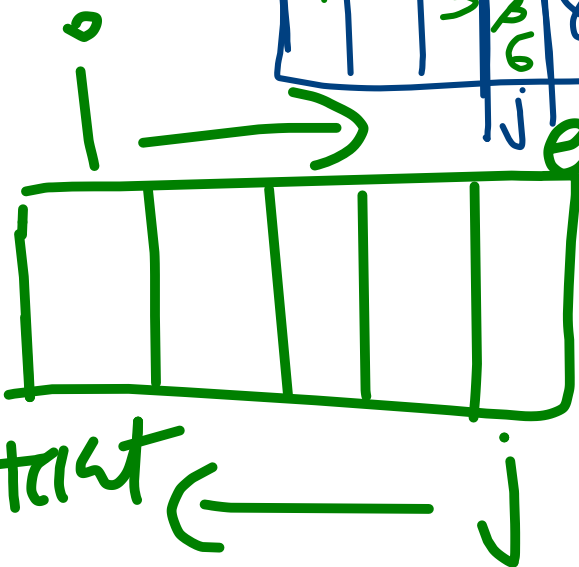
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

The image illustrates the selection sort algorithm through three sequential steps, each showing an array and the process of finding and swapping the minimum element.

Step 1: The initial array is $[10, 2, 8, 1, 6]$ with indices $0, 1, 2, 3, 4$. The minimum element 1 at index 3 is identified and swapped with the element at index 0 (10).

Step 2: The array is now $[1, 2, 8, 6, 10]$. The minimum element 2 at index 1 is identified and swapped with the element at index 3 (6).

Step 3: The array is now $[1, 2, 6, 8, 10]$. The minimum element 6 at index 2 is identified and swapped with the element at index 3 (8).

~~| | | | | | |
|----|---|---|---|---|----|
| 10 | 2 | 8 | 1 | 6 | 3 |
| 3 | | | | | 10 |~~
~~| | | | | | |
|---|---|---|---|---|----|
| 3 | 2 | 8 | 1 | 6 | 10 |
| | | 6 | | 8 | |~~
~~| | | | | | |
|---|---|---|---|---|----|
| 3 | 2 | 6 | 1 | 8 | 10 |
| 1 | | 3 | 2 | | |~~


```

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

```

Q(a,0,5)

10	2	8	6	3	
3	3			15	
0	1	2	3	4	5
i	j				

i: 0 < 5

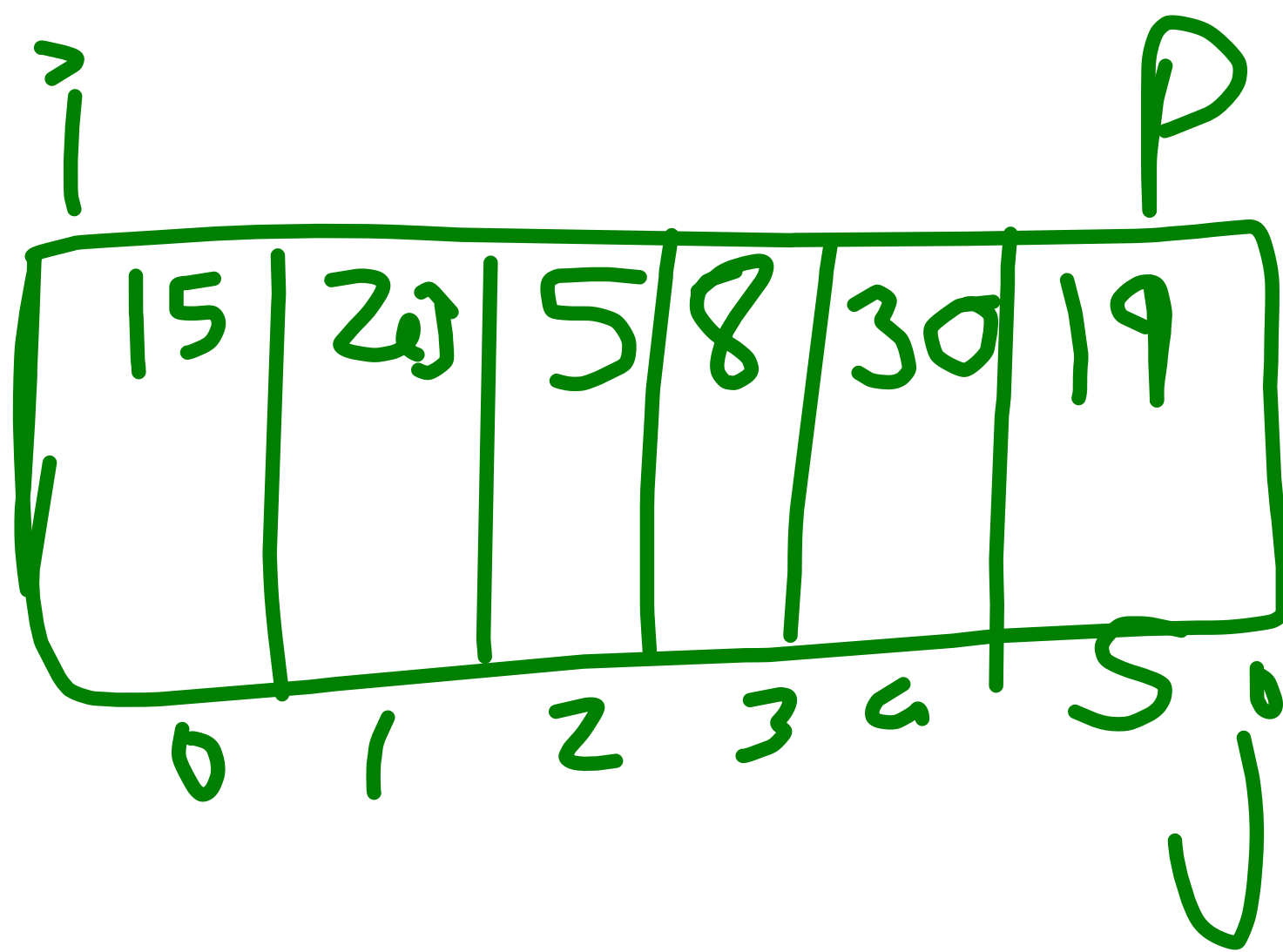
Qs(a,1,5)

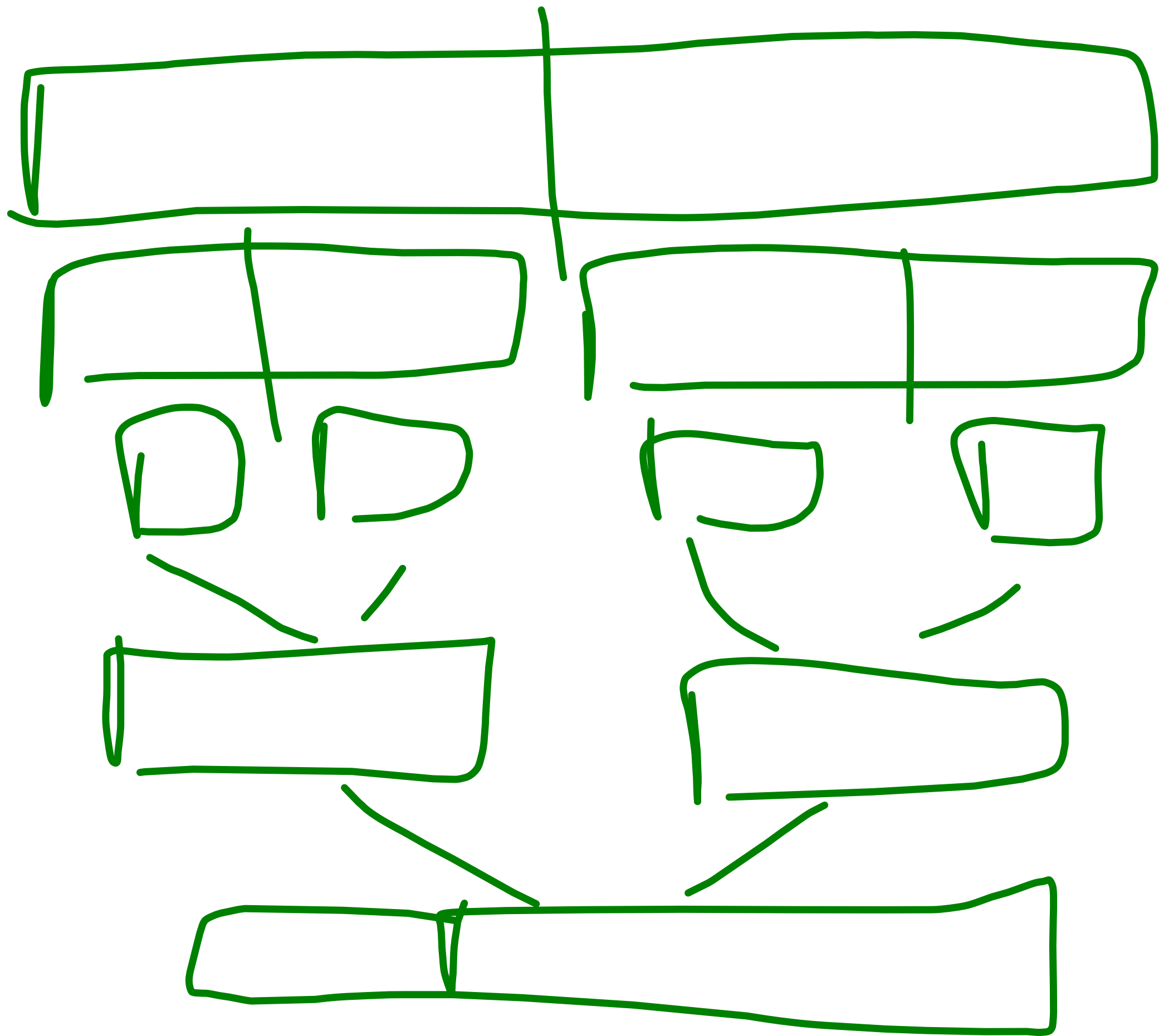
2	3	10	8	6	15
i	j				
p					

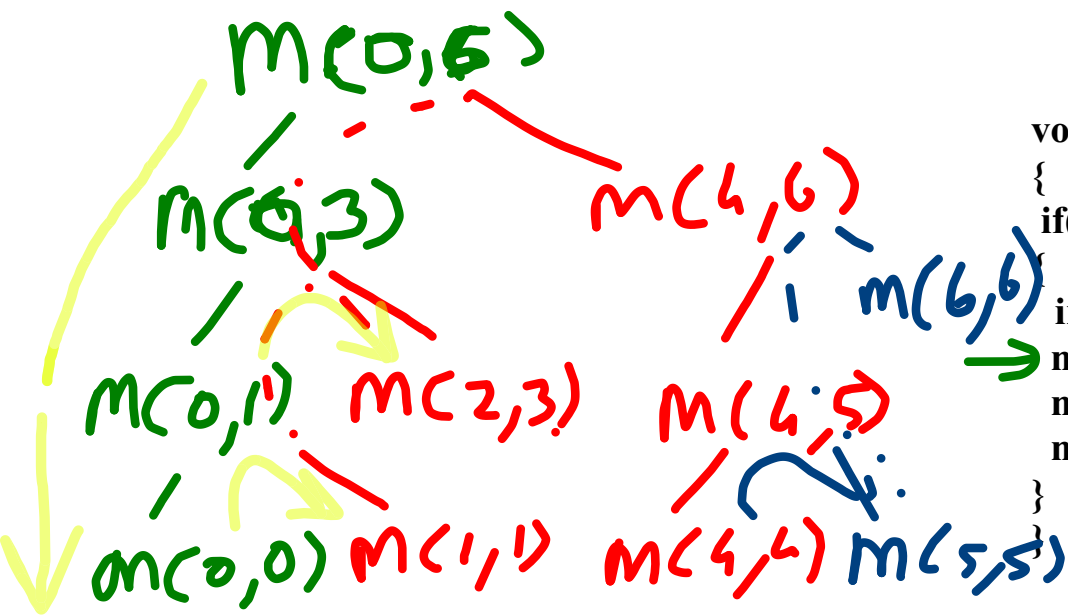
```

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

```

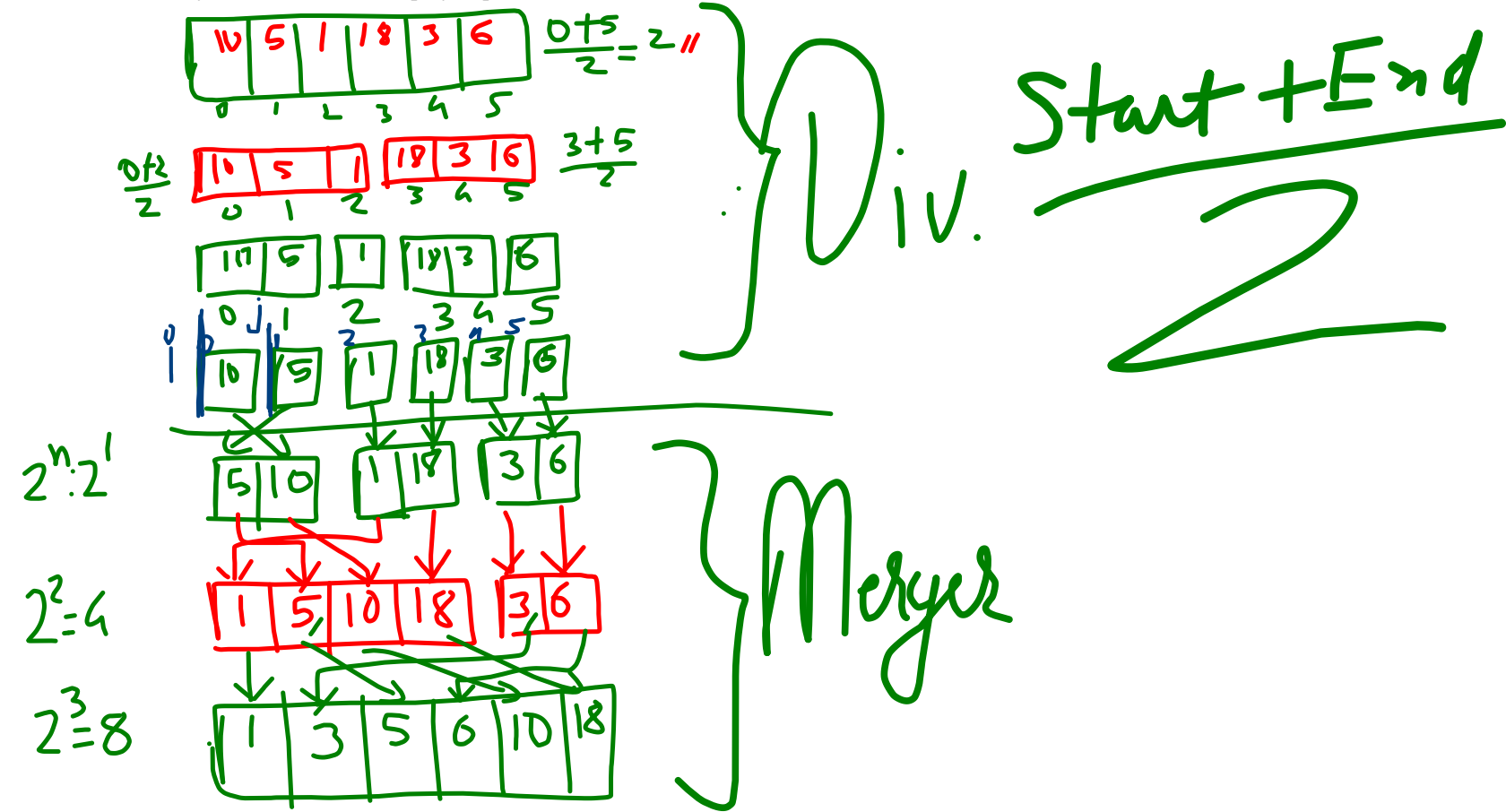


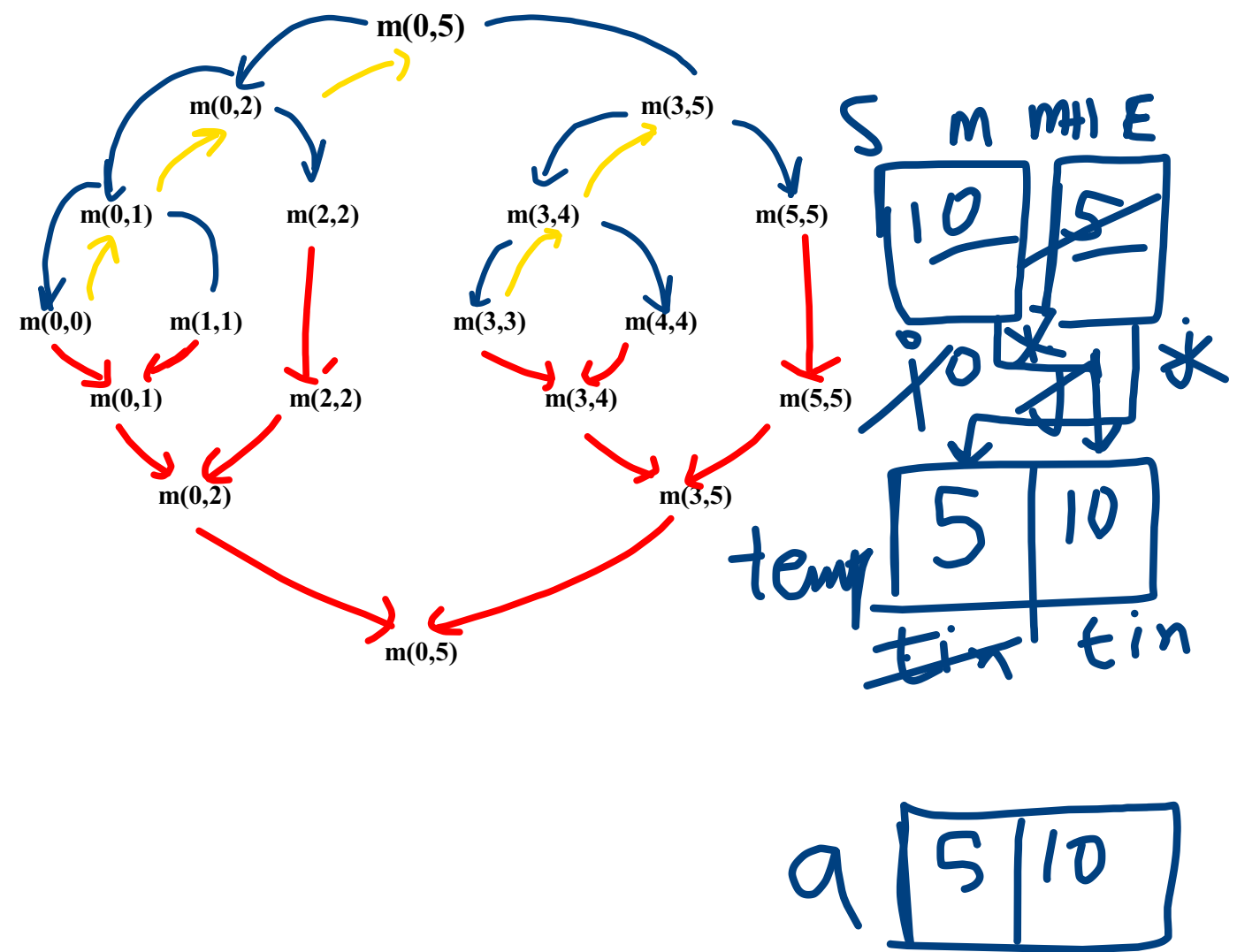




```
void merge_sort(int a[], int start, int end)
{
    if (start < end)
    {
        int mid = (start + end) / 2;
        merge_sort(a, start, mid);
        merge_sort(a, mid + 1, end);
        merge(a, start, mid, end);
    }
}
```

Merge Sort:
 in this we start with 1 array of n elements
 divide it from mid again and again till we get n array of size 1. Then merge this
 arrays in sorted manner step by step





```
void merger(int a[],int start,int mid,int end)
{
    int i,j,t[],tindex;
    i=start;
    j=mid+1;
    //temp array to keep sorted data
    t=new int(a.length);
    tindex=start;
    //compare and copy in order
    while(i<=mid && j<=end)
    {
        if(a[i]<a[j])
            temp[tindex++]=a[i++];
        else
            temp[tindex++]=a[j++];
    }
    //copy leftover
    while(i<=mid)
        tmpe[tindex++]=a[i++];
    while(j<=end)
        tmpe[tindex++]=a[j++];

    //copy back to a from temp
    for(i=start;i<=end;i++)
        a[i]=temp[i];
}
```

