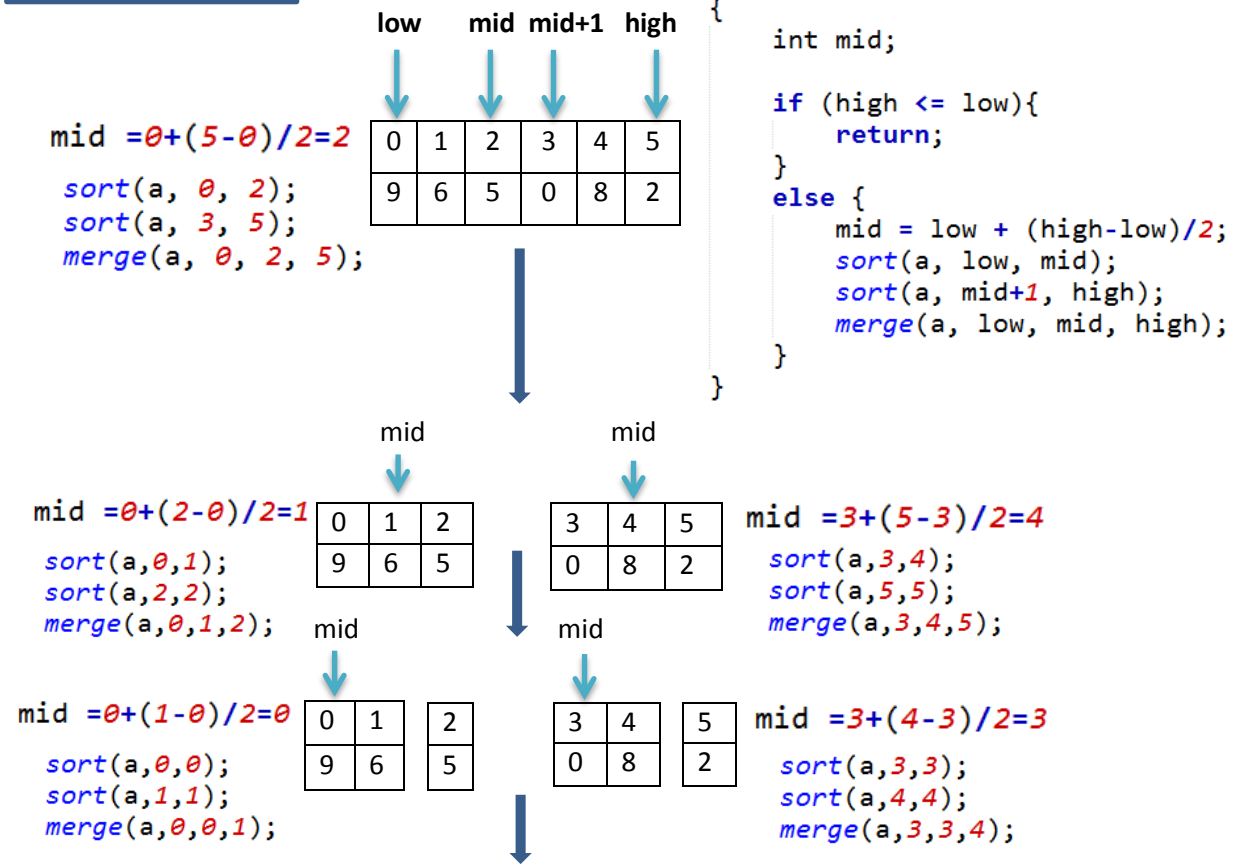
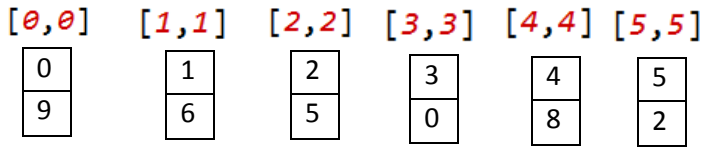
MERGE SORT

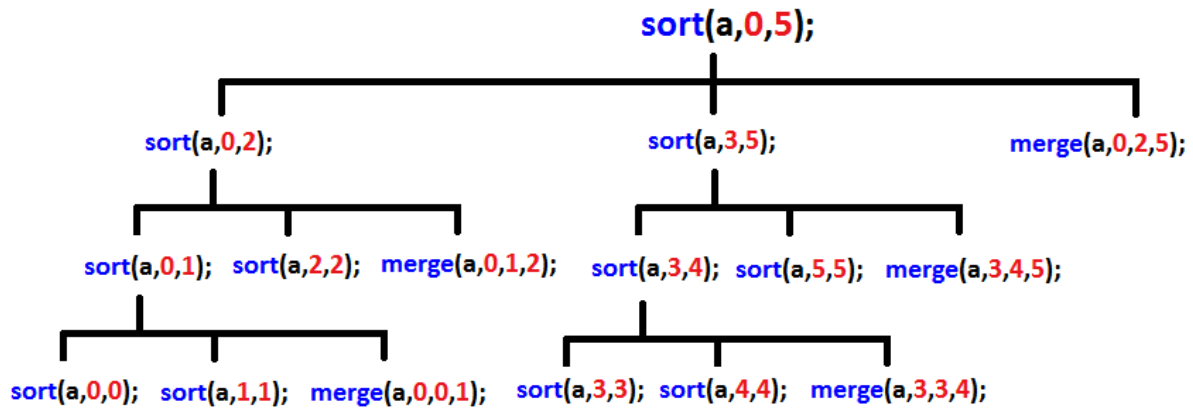
```
sort(a, 0, 5);
```

a [9, 6, 5, 0, 8, 2]

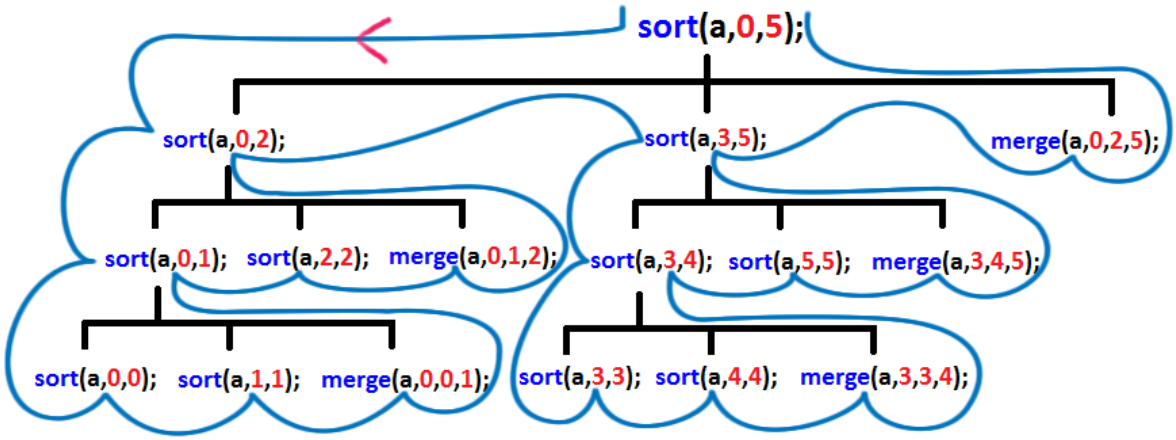
```
int sort(int a[],int low,int high)
{
    int mid;

    if (high <= low){
        return;
    }
    else {
        mid = low + (high-low)/2;
        sort(a, low, mid);
        sort(a, mid+1, high);
        merge(a, low, mid, high);
    }
}
```







Merge Sort:



0	1	2	3	4	5
9	6	5	0	8	2

N.B. Single Element is always sorted.

Step 1:

- In order to sort 0 to 5 elements of array **a** i.e. **a[9, 6, 5, 0, 8, 2]**, sort function **sort(a,0,5)** calls **sort(a,0,2)** which means the program needs to sort index **0 to 2** that is elements **[9, 6, 5]** first.

Step 2:

- **sort(a,0,2)** function **calls sort(a,0,1)** which means to sort index **0 to 2,** elements [9, 6, 5], the program needs to sort index **0 to 1,** element [9, 6] first.

Step 3:

- Now **sort(a,0,1)** function calls **sort(a,0,0)**. Which means to sort index **0 to 1,** elements [9, 6], the program needs to sort index **0**, element [9] first. As we know single element is already sorted so [9] element is sorted.

Step 4:

- Then program calls the next function **sort(a,1,1)**, which is also sorted as it contain single element [6].

Step 5:

- Then the program calls function **merge(a,0,0,1)** which means the function will merge **0 to 0** element with **1 to 1** element. Which is basically index **0,** with element **[9]** and index **1**, with element **[6]**. After merging new value will be [6, 9].

0	1
6	9

Step 6:

- Now element of array index 0 to 1 is sorted. That means **sort(a,0,1)** is done. Next the program calls function **sort(a,2,2)** which is also sorted as it contain single element [5].

Step 7:

- Then the **merge(a,0,1,2)** function is called. **(a,0,1,2)** means, to merge index **0 to 1, element [6,9]** with index **2 to 2, element [5]**. 0 to 1 is already sorted in step 5 and value of the sort(a,0,1) is [6,9]. So after merging [6,9] and [5] new sorted element will be [5, 6, 9]. Now we can see **sort(a,0,2)** is completed and so far we have sorted and merged the left half that is 0 to 2 element of the array **a**.

Step 8:

0	1	2
5	6	9

- Now the program will call **sort(a,3,5)** to sort index **3 to 5,** element **[0, 8, 2]** which is the right half of the array **a**.

3	4	5
0	8	2

Step 9:

- **sort(a,3 5)** function calls **sort(a,3,4)** which means to sort index **3 to 5,** elements **[0 , 8, 2]**, the program needs to sort index **3 to 4**, elements **[0, 8]** first.

Step 10:

- Now **sort(a,3,4)** function calls **sort(a,3,3)**. Which means to sort index **3 to 4,** elements [0, 8] the program needs to sort index **3**, element **[0]** first.

Step 11:

- We know that single element is already sorted. So index **3**, element [0] is sorted. That is sort(a, 3, 3) is sorted and it's done.

Step 12:

- Next **sort(a,4,4)** is called and again it's sorted because it contains single element [8].

Step 13:

- Now the **merge(a,3,3,4)** function is called. Which means the function will merge index **3 to 3** with index **4 to 4**. That is index 3 with element [0] and index 4 with element [8]. After merging new value will be [0, 8].

Step 14:

- Now we see **sort(a,3,4)** is completed and sorted. So the program calls the next function **sort(a,5,5)**. Which is also sorted as it contains single element [2].

Step 15:

- Then **merge(a,3,4,5)** function is called. Which will merge index **3 to 4**, element **[0,8]** with index **5 to 5**, element **[2]**. After merging new sorted value will be **[0, 2, 8]**.

3	4	5
0	2	8

- So now **sort(a,3,5)** function is completed and sorted with element [0,2,8]. And previously **sort(a,0,2)** function was also sorted with element **[5,6,9]**

sort(a,0,2);

0	1	2
5	6	9

sort(a,3,5);

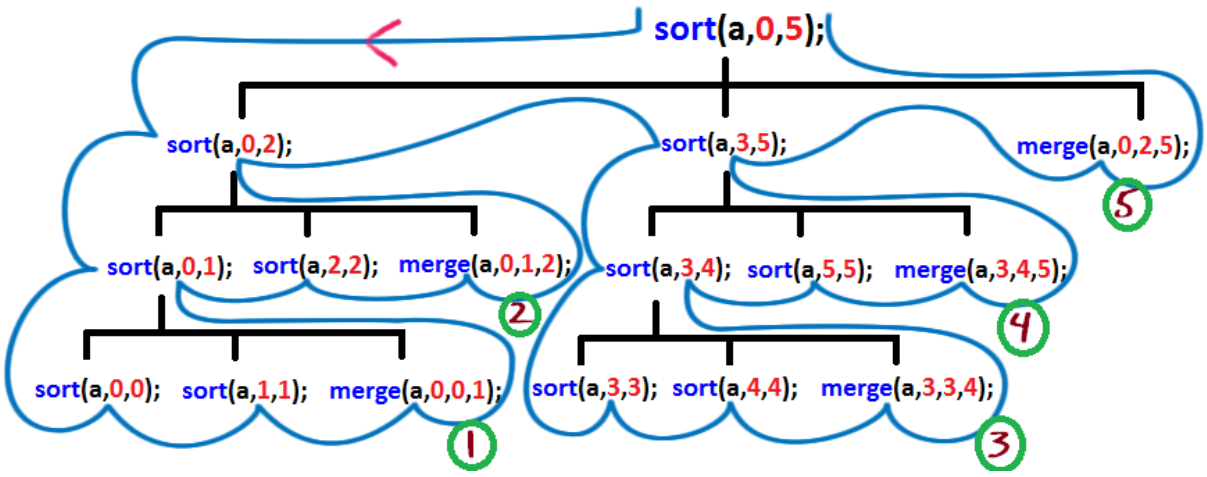
3	4	5
0	2	8

Step 16:

- Then **merge(a,0,2,5)** will be called to merge index **0 to 2**, element [5, 6, 9] with index **3 to 5**, element [0, 2, 8]. After merging the sorted element will be **[0, 2, 5, 6, 8, 9].** Finally **sort(a,0,5)** function is completed and array a is sorted.

0	1	2	3	4	5
0	2	5	6	8	9

Merge Function Code:



This method merges by first copying elements into the auxiliary array aux[]. Then merging back to a[]. In the merge (the second for loop), there are four conditions:

1. Left half exhausted (take from the right),
2. Right half exhausted (take from the left),
3. Current key on right less than current key on left (take from the right),
4. Current key on right greater than or equal to current key on left (take from the left).

```
int merge(int a[], int low, int mid, int high)
{
     int i = low, j = mid+1, k;
     int aux[6];

     for (k = low; k <= high; ++k){
          aux[k] = a[k];
     }

     for (k = low; k <= high; ++k){

          if (i > mid){
               a[k] = aux[j];
               j++;
          }
          else
               if (j > high ){
                    a[k] = aux[i];
                    i++;
               }
          else
               if ((aux[j]<aux[i])){
                    a[k] = aux[j];
                    j++;
               }
          else {
               a[k] = aux[i];
               i++;
          }
     }
}
```

First merge function call is **merge(a,0,0,1)**. That means merge function receiving arguments **low = 0, mid = 0, high =1.**

First for loop:

When k =0, aux[0] = 9

When k =1, aux[1] = 6

Result:

- **aux[9,6]**

We have copied sorted 0 and 1 element of array a[] to array aux[].

Second for loop:

When k =0,

```
if((aux[1]<aux[0])){
    a[0] = aux[1];
    j++;
}
```

Result:

- **a[6]**
- **j = 2.**

When k =1,

```
if (j > high ){
  a[1] = aux[0];
  i++;
}
```

Result:

- **a[6,9]**
- **i = 1**

So **merge(a,0,0,1)** is completed and **a[9,6]** is sorted now as **a[6,9]** thus **sort(a,0,1)** is sorted.

Main Function:

```
#include<stdio.h>

int sort(int a[], int low, int high);
int merge(int a[], int low, int mid, int high);

int main(void)
{
    int size =6;
    int i;
    int a[] = {9,6,5,0,8,2};

    sort(a, 0, size-1);

    for(i=0; i<size; ++i)
    {
        printf("%3d", a[i]);
    }
    puts("");
}
```