Lecture 09

Md. Shahid Uz Zaman

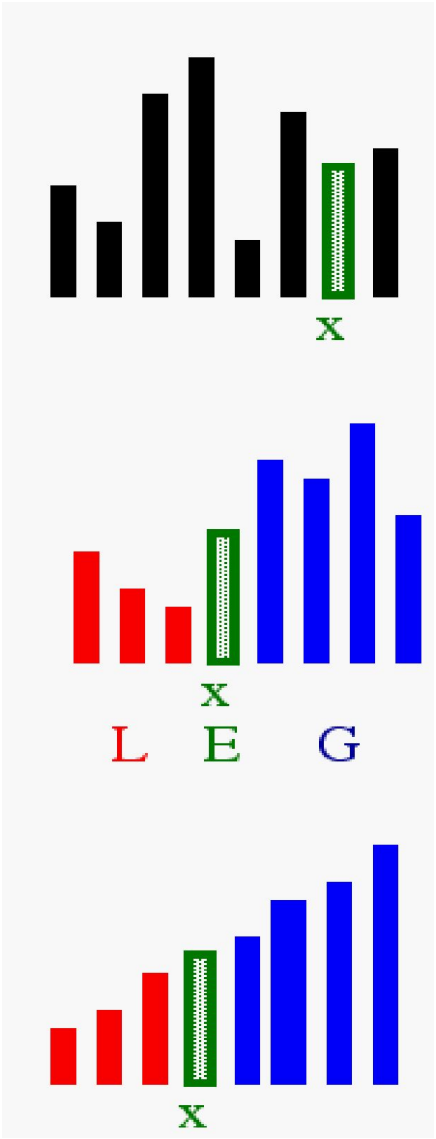
Quick Sort

Idea of Quick Sort

1) **Select**: pick an element

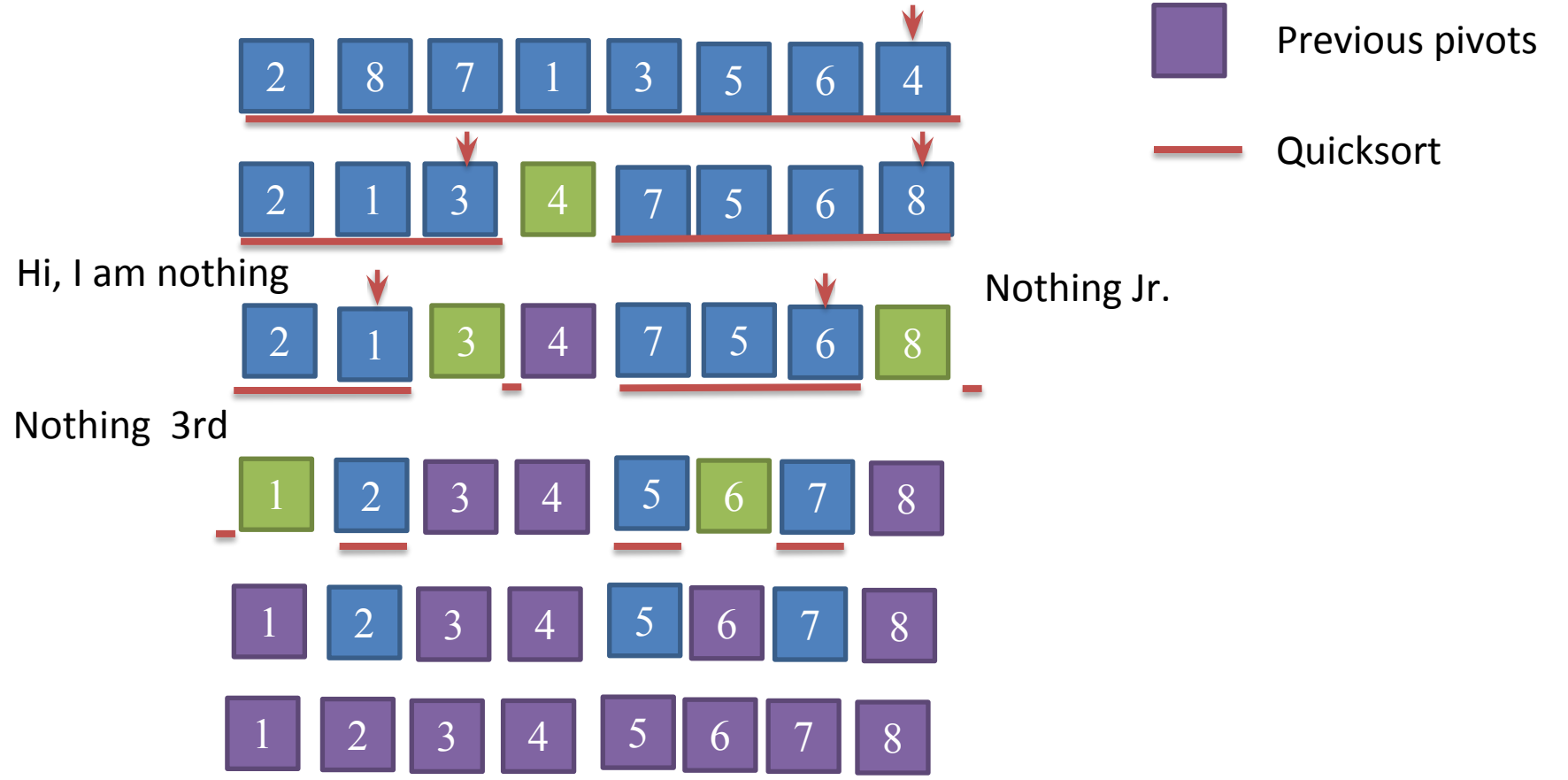
2) **Divide**: rearrange elements so that x goes to its final position E

3) **Recurse and Conquer**: recursively sort



Quick Sort: Algorithm

- Quicksort example



Quick Sort: Example

Example of Quick Sort:

44 33 11 55 77 90 40 60 99 22 88

Let **44** be the **Pivot** element and scanning done from right to left

Comparing **44** to the right-side elements, and if right-side elements are **smaller** than **44**, then swap it. As **22** is smaller than **44** so swap them.

22 33 11 55 77 90 40 60 99 **44** 88

Now comparing **44** to the left side element and the element must be **greater** than 44 then swap them. As **55** are greater than **44** so swap them.

22 33 11 **44** 77 90 40 60 99 **55** 88

Recursively, repeating steps 1 & steps 2 until we get two lists one left from pivot element **44** & one right from pivot element.

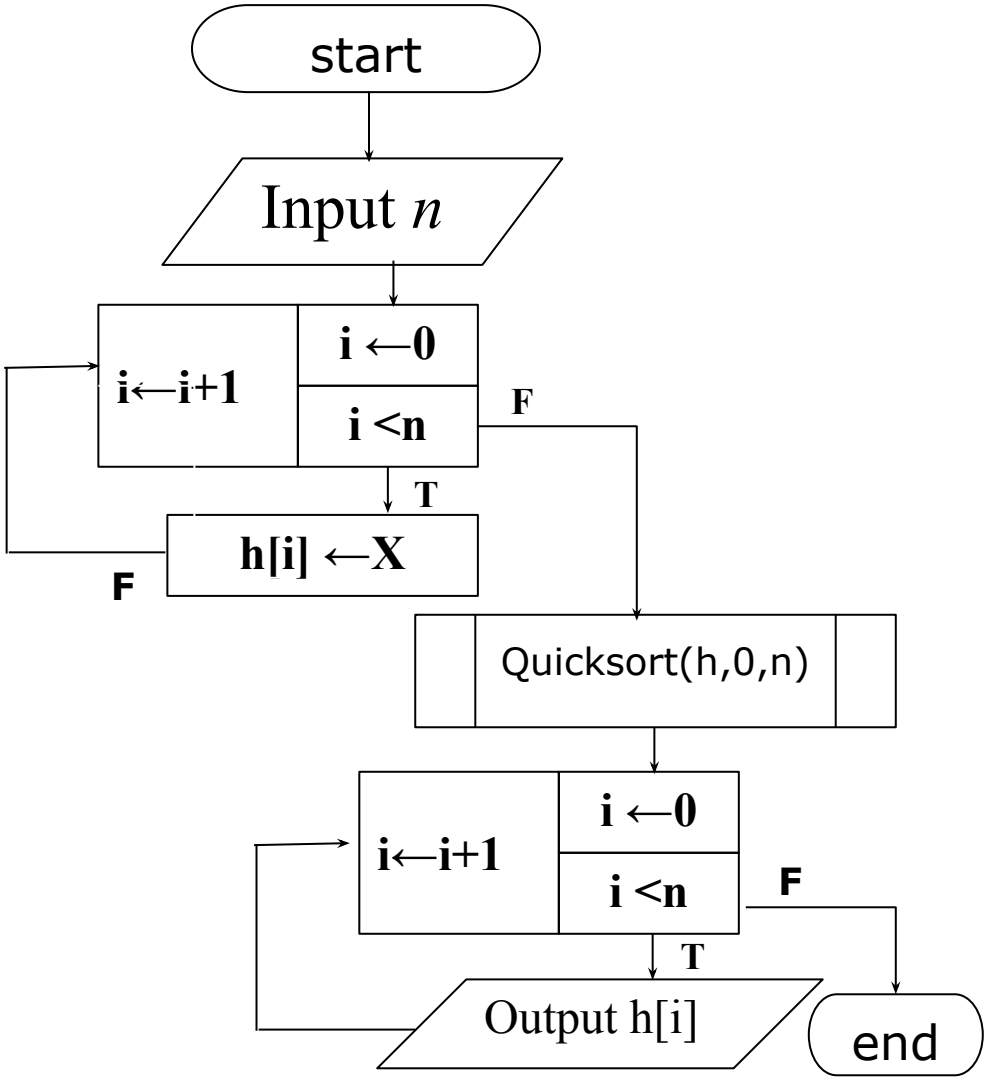
22 33 11 **40** 77 90 **44** 60 99 55 88

Swap with 77:

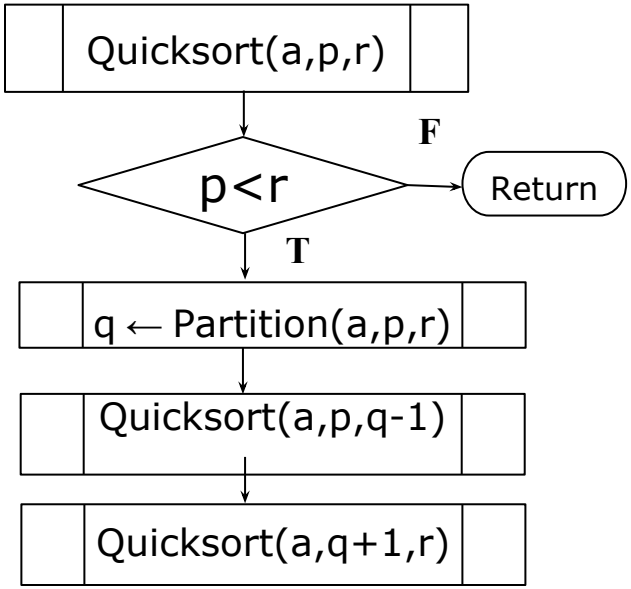
22 33 11 40 **44** 90 **77** 60 99 55 88

Now, the element on the right side and left side are greater than and smaller than **44** respectively.

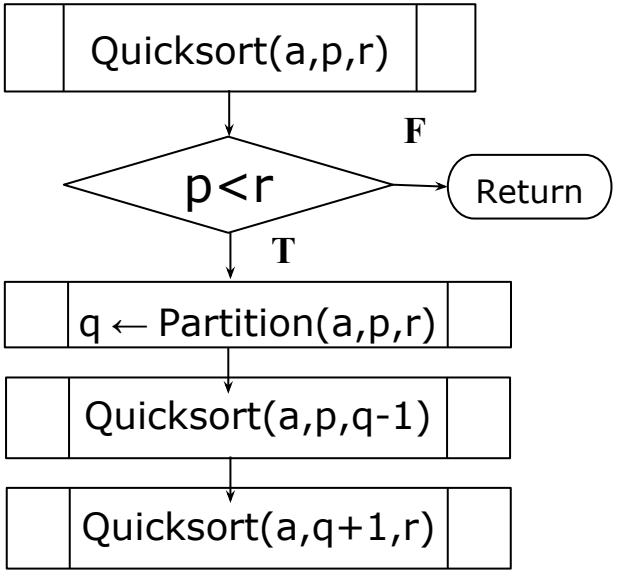
Quick Sort: Algorithm



Flowchart



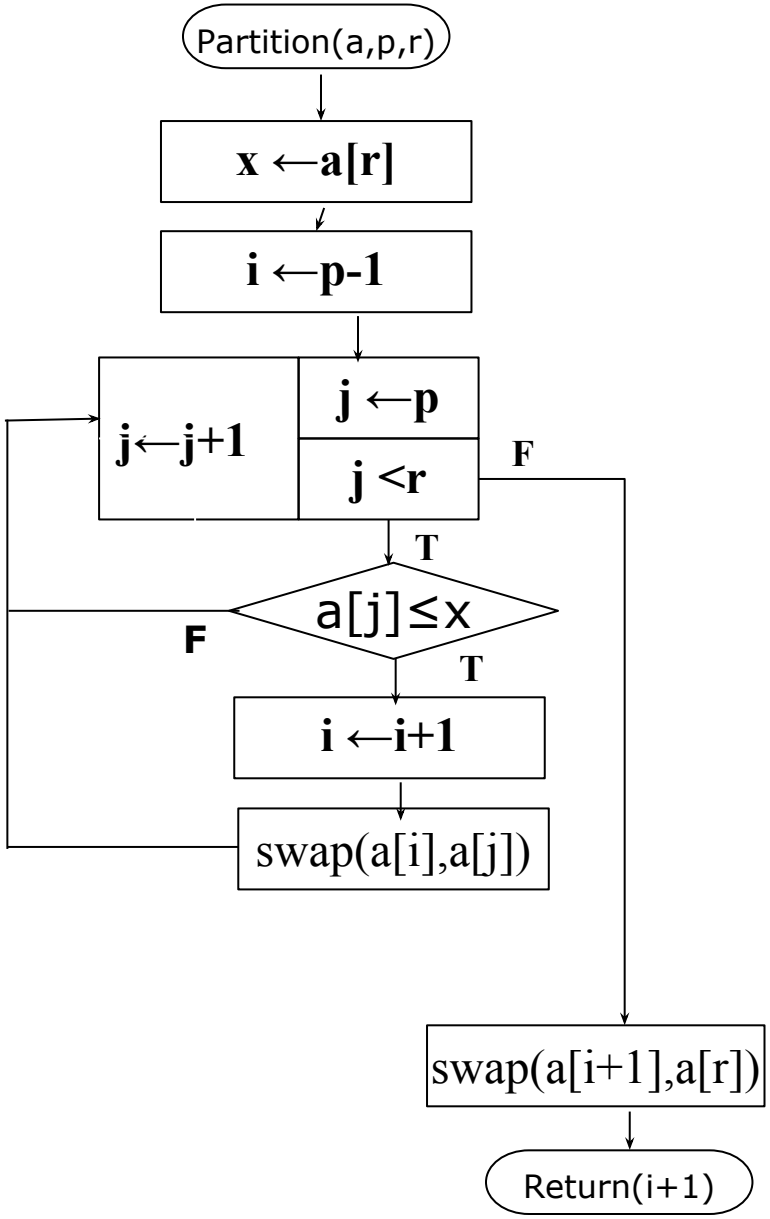
Quick Sort: Algorithm



a[6]={5,7,6,1,3,2,4}

i: puts elements<x at left

j: search all element except x

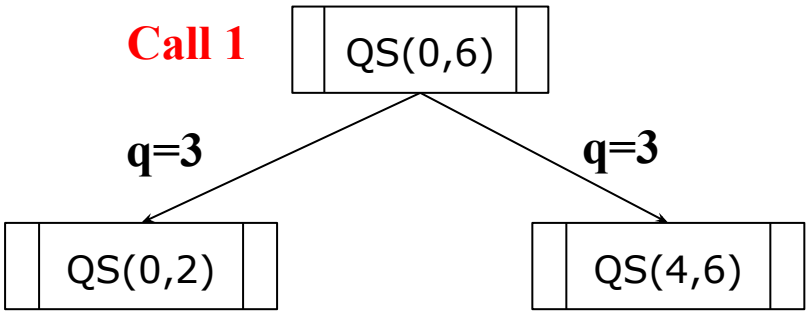


Flowchart

Quick Sort: Algorithm

Function Calling

a[7]={5,7,6,1,3,2,4}

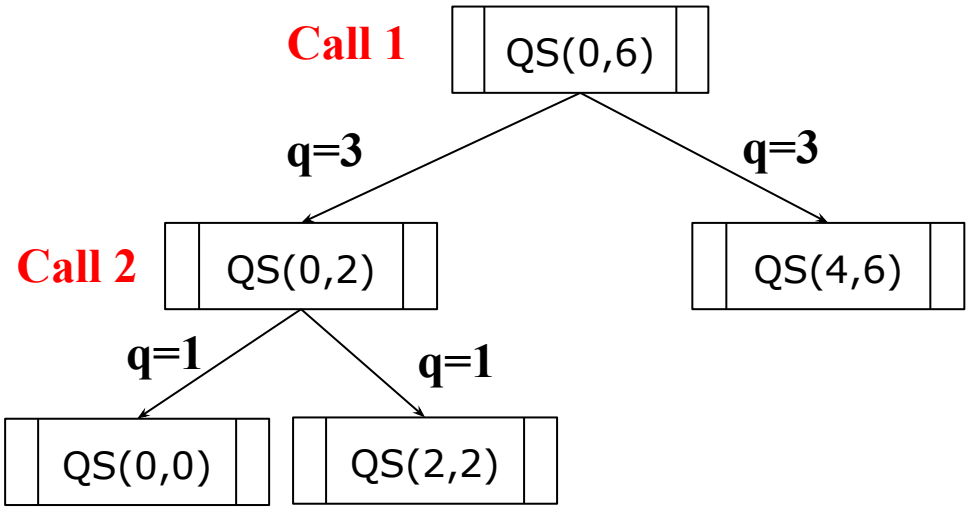


Range
Left: p - (q-1)
Right: (q+1) - r

Quick Sort: Algorithm

Function Calling

a[7]={5,7,6,1,3,2,4}

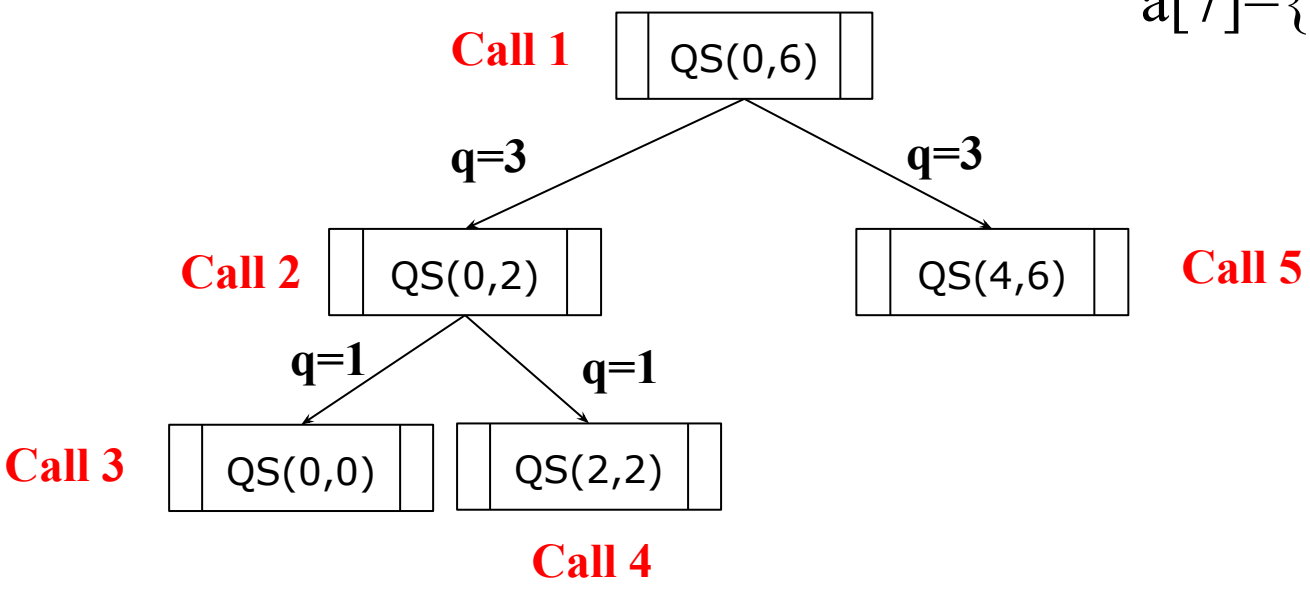


Range
Left: p - (q-1)
Right: (q+1) - r

Quick Sort: Algorithm

Function Calling

a[7]={5,7,6,1,3,2,4}

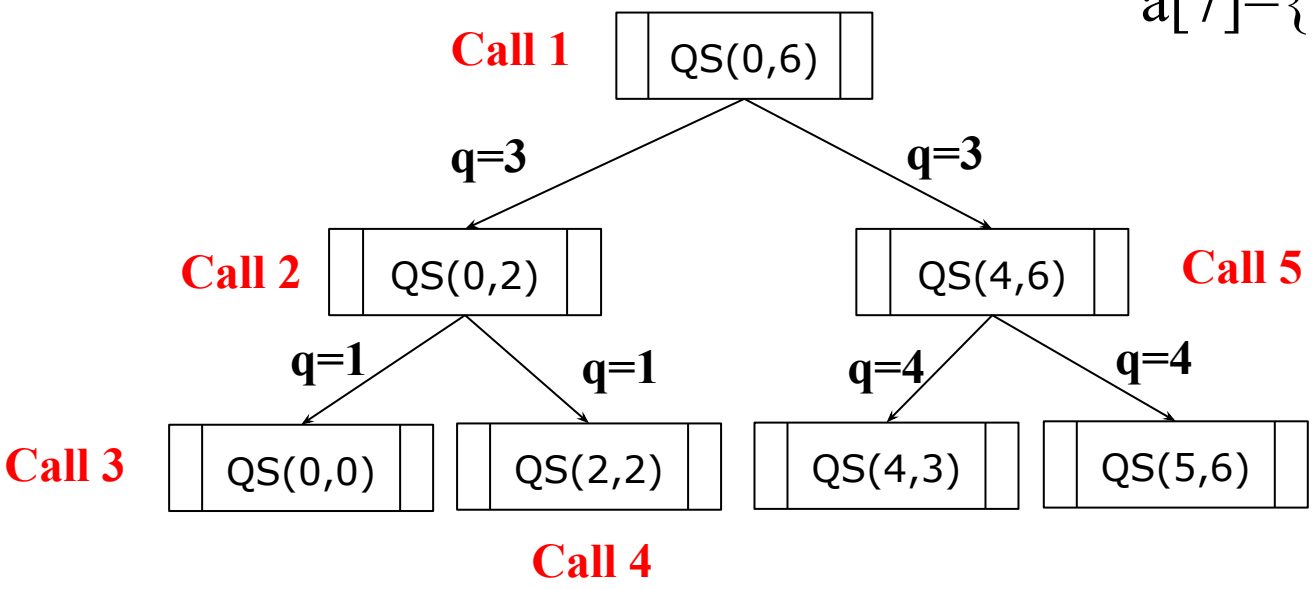


Range
Left: p - (q-1)
Right: (q+1) - r

Quick Sort: Algorithm

Function Calling

a[7]={5,7,6,1,3,2,4}

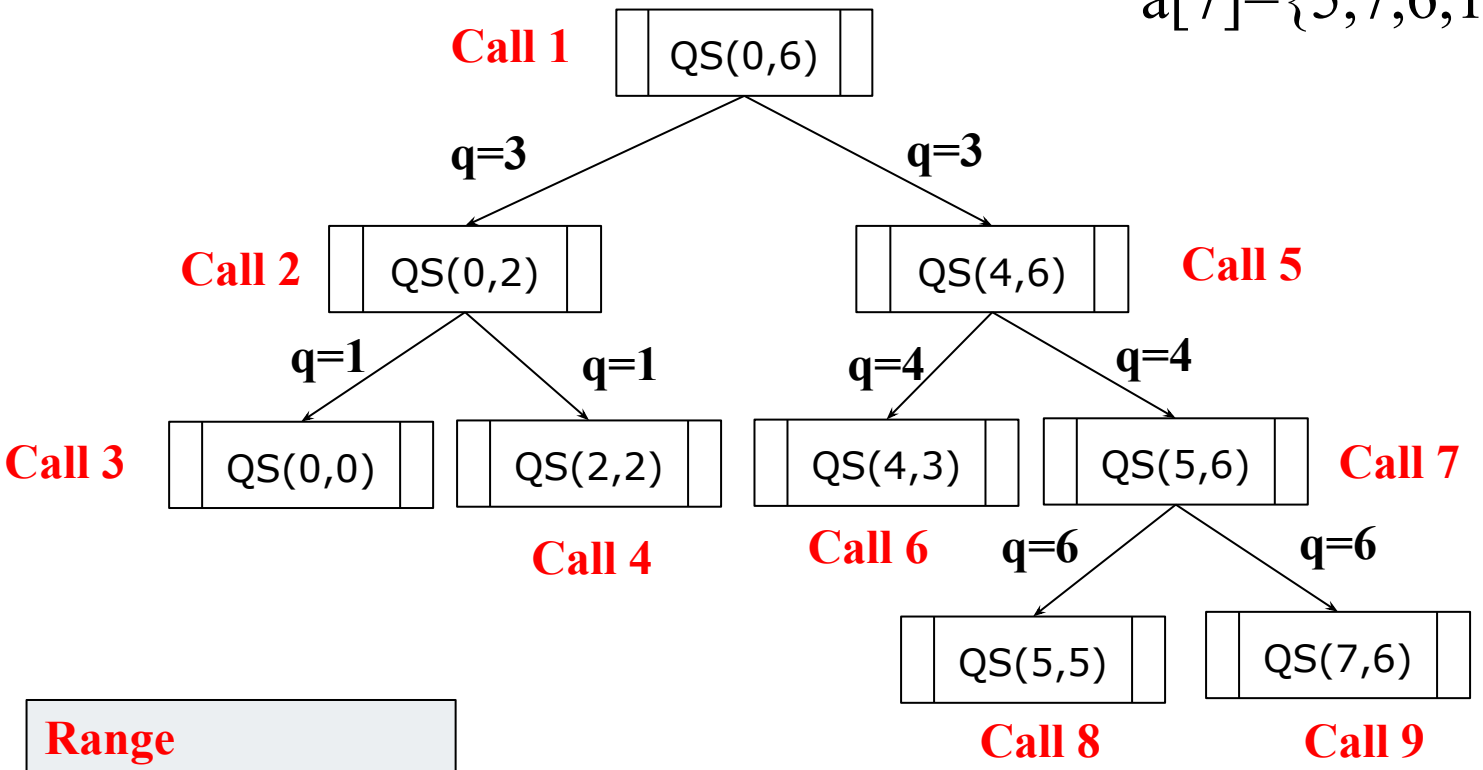


Range
Left: p - (q-1)
Right: (q+1) - r

Quick Sort: Algorithm

Function Calling

a[7]={5,7,6,1,3,2,4}



Range
Left: p - (q-1)
Right: (q+1) - r

Quick Sort: Algorithm

```
#include <stdio.h>
#include <stdlib.h>
void QuickSort(int *a,int p,int r);
int Partition(int *a,int p,int r);
int main()
{
   int a[7]={5,7,6,1,3,2,4};
   int p,r,i;
   p=0;r=6;
   QuickSort(a,p,r);
   for(i=p;i<=r;i++)
   printf("%d  ",a[i]);
   return 0;
}
```

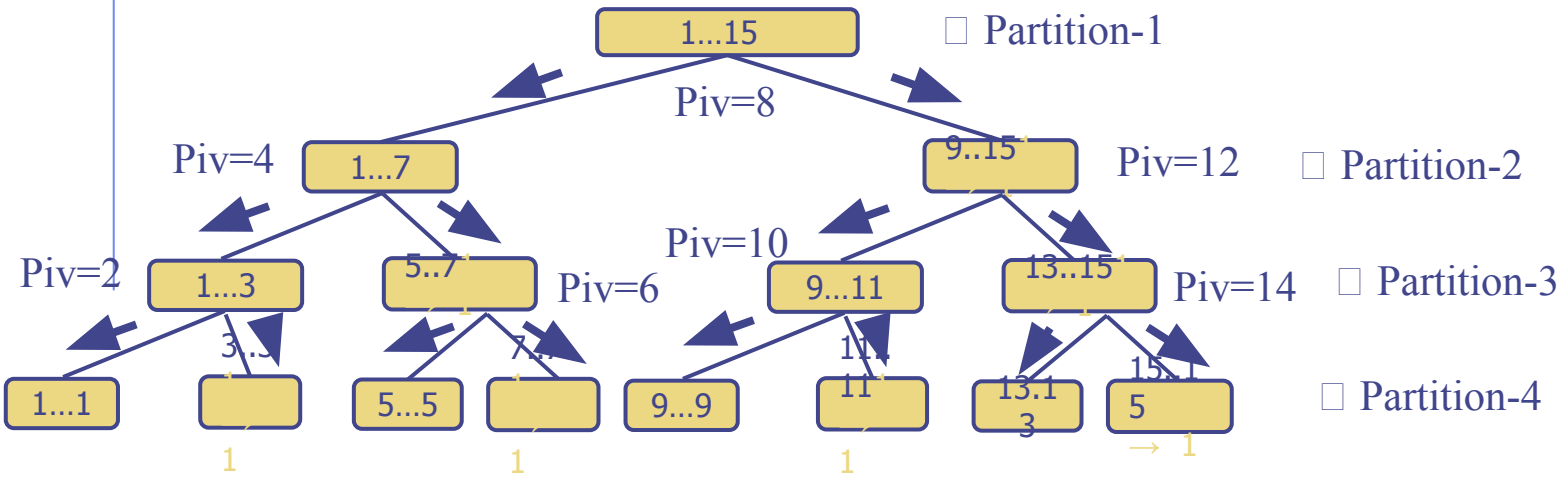
```
void QuickSort(int *a,int p,int r){
int i,q;
   cout<<"p="<<p<<"
r="<<r<<endl;
   if(p<r){
   q=Partition(a,p,r);
   cout<<"calling 1
q="<<q<<endl;
   QuickSort(a,p,q-1);
   cout<<"calling 2
q="<<q<<endl;
   QuickSort(a,q+1,r);
   //cout<<" q="<<q<<endl;
   }
   return;
}
```

```
int Partition(int *a,int p,int r){
int x,j,t,i;
x=a[r]; i=p-1;
cout<<"at partition x="<<x<<"
i="<<i<<endl;
for(j=p;j<=r-1;j++){
   if(a[j]<=x){
      i++;

cout<<"swap("<<a[i]<<","<<a[j
]<<")"<<endl;
      t=a[i];a[i]=a[j];a[j]=t;
   }
}
t=a[i+1];a[i+1]=a[r];a[r]=t;

cout<<"swap("<<a[i+1]<<","<<
a[r]<<")"<<endl;
cout<<"From partition
q="<<i+1<<endl;
return(i+1);
}
```

Quick Sort: Algorithm

Expected Running Time

- Consider an array with 15 elements
  - **Best Case:** Pivot always places at the middle of the list



No of elements in each sub-list after 1st Partition=n/2
No of elements in each sub-list after 2nd Partition=n/2/2
No of elements in each sub-list after kth Partition=$n/2^k$

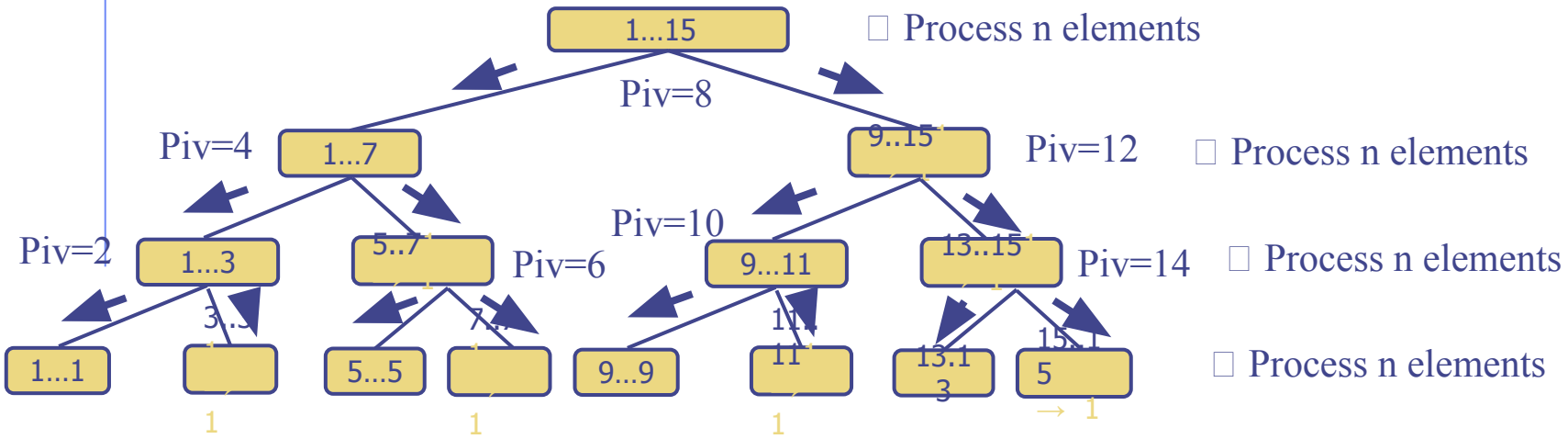
Finally, there will be only one element per sub-list

- $n/2^k=1$
- $k=\log_2 n$

Quick Sort: Algorithm

Expected Running Time

- Consider an array with 15 elements
  - **Good call:** Pivot always places at the middle of the list



It is a binary tree with level k

$k=\log_2 n$

In each level n elements are processed

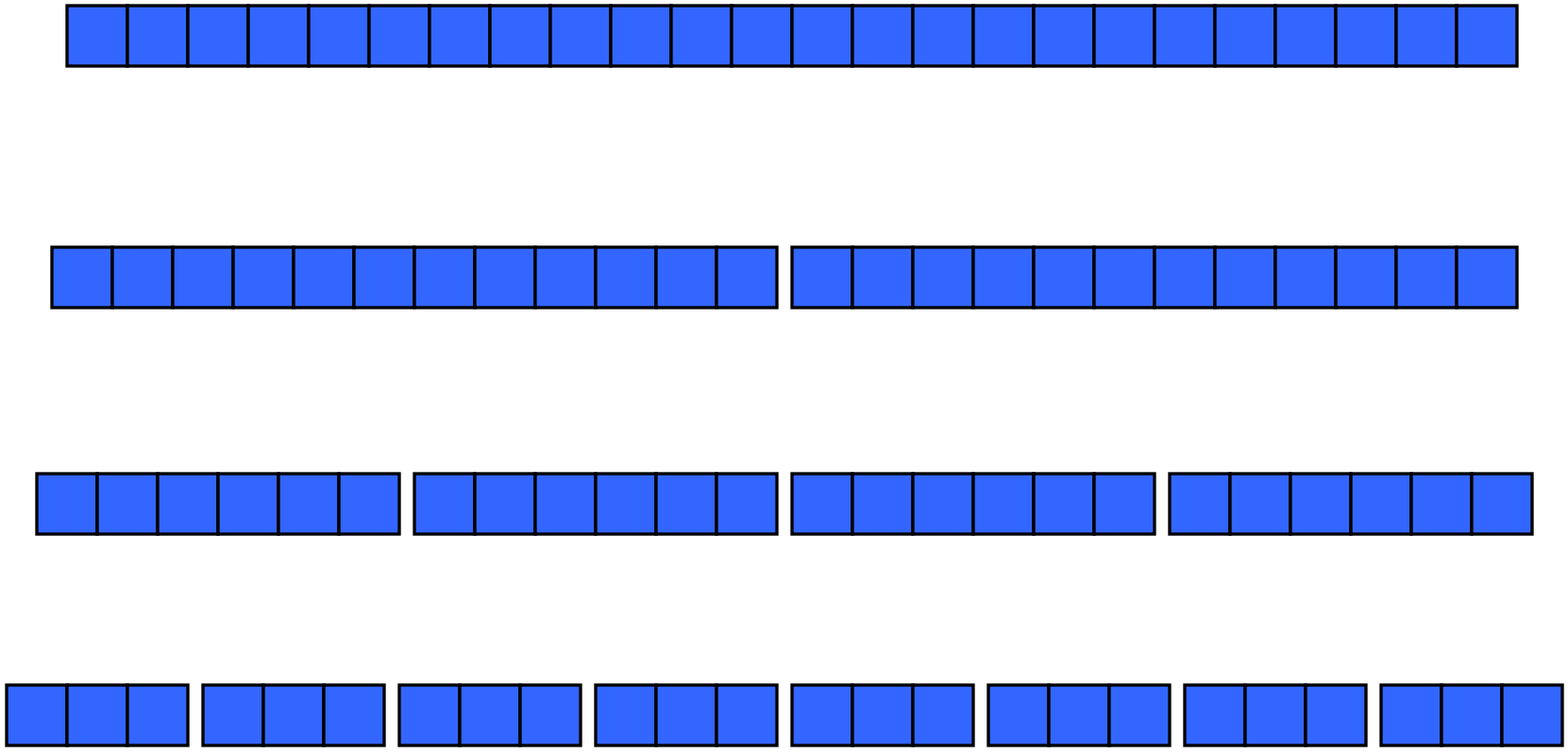
So Time Complexity= $O(n\log_2 n)$

Select median as pivot

Practically impossible

Quick Sort: Algorithm

Partitioning at various levels



Worst-case Running Time

- The worst case for quick-sort occurs when the pivot is the unique minimum or maximum element
- One of L and G has size $n - 1$ and the other has size 0
- The running time is proportional to the sum

$$n + (n - 1) + \ldots + 2 + 1$$

- Thus, the worst-case running time of quick-sort is $O(n^2)$