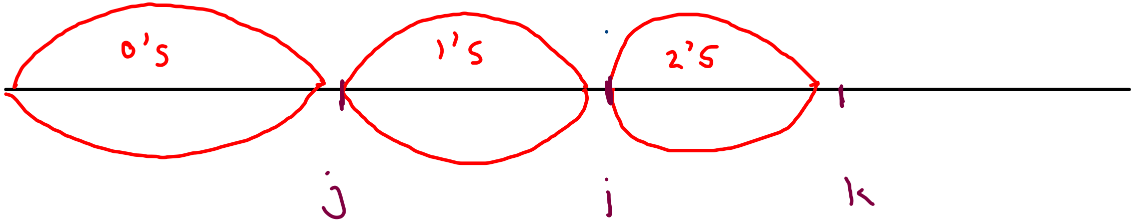
# Sort 012

0 1 0 0 2 0 1 1 2 1



(i) linear

(ii) single traversal

(iii) no extra space

0 to j-1 -> 0's

j to i-1 -> 1's

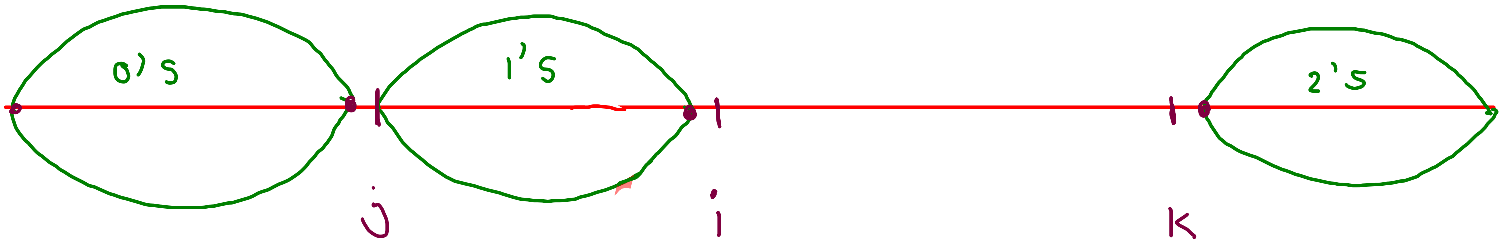
i to k-1 -> 2's

k to end -> unk

| a[k]==2 | a[k]==1 | a[k]==0 |
|---|---|---|
| k++. | swap(i,k);<br>i++;<br>k++; | swap(j,k);<br>swap(i,k);<br>j++; i++; k++; |

(edge case)

# Sort 012



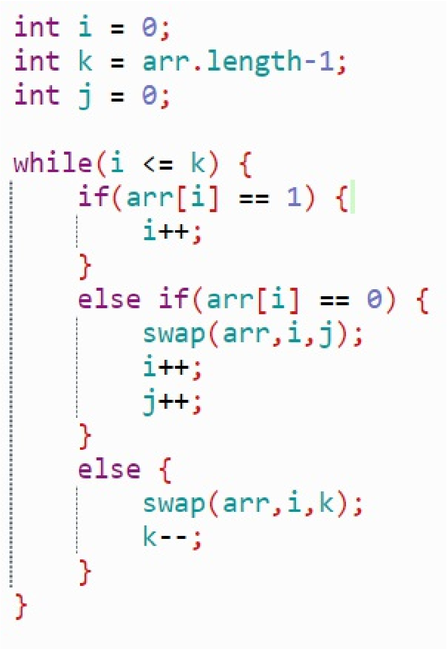
| a[i] == 1 | a[i] == 0 | a[i] == 2 |
|---|---|---|
| i++; | swap(i,j);<br>i++;<br>j++; | swap(i,k);<br>k--; |

0's → 0 to j-1

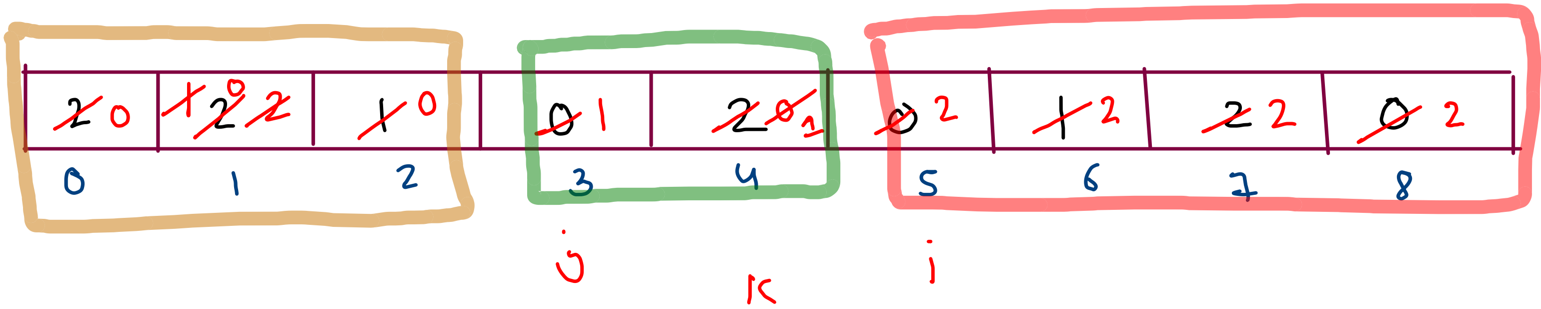
1's → j to i-1

2's → k+1 to end

Uk → i to k



```
int i = 0;
int k = arr.length-1;
int j = 0;

while(i <= k) {
    if(arr[i] == 1) {
        i++;
    }
    else if(arr[i] == 0) {
        swap(arr,i,j);
        i++;
        j++;
    }
    else {
        swap(arr,i,k);
        k--;
    }
}
```

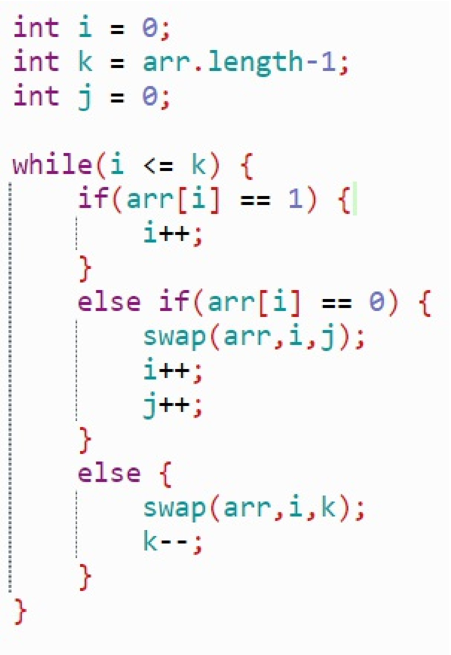


0's → 0 to j-1

1's → j to i-1

2's → k+1 to end

Uk → i to k



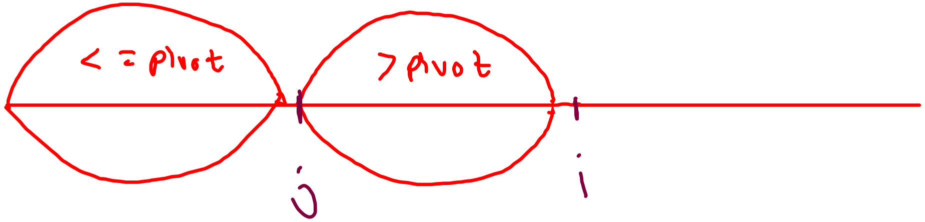
```
int i = 0;
int k = arr.length-1;
int j = 0;

while(i <= k) {
    if(arr[i] == 1) {
        i++;
    }
    else if(arr[i] == 0) {
        swap(arr,i,j);
        i++;
        j++;
    }
    else {
        swap(arr,i,k);
        k--;
    }
}
```

| | | | | | |
|---|---|---|---|---|---|
| ~~2~~ 0 | 0 | 2 | ~~0~~ 2 | 2 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 |
| | k | i<br>j | | | |

# Partition An Array

-2 4 6 8 3 9 6

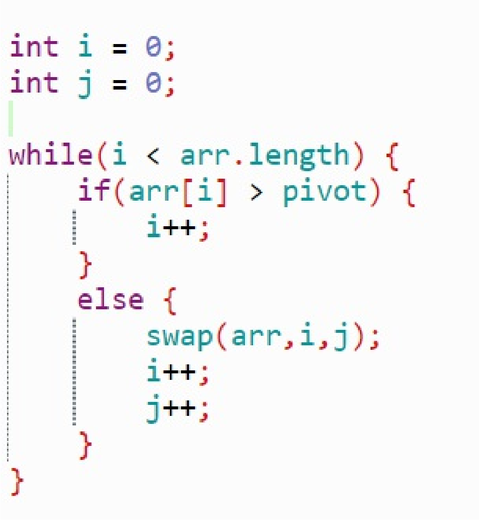
pivot



0 to j-1 -> <= pivot

j to i-1 -> > pivot

```
if (a[i] > pivot) {
    i++;
}
else {
    swap(i,j);
    i++; j++;
}
```



```
int i = 0;
int j = 0;

while(i < arr.length) {
    if(arr[i] > pivot) {
        i++;
    }
    else {
        swap(arr,i,j);
        i++;
        j++;
    }
}
```

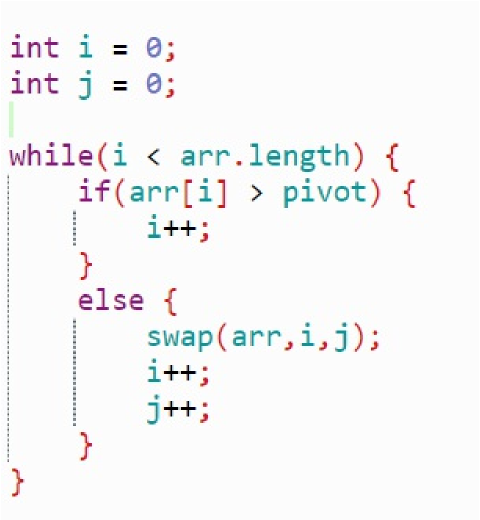
pivot = 6

6

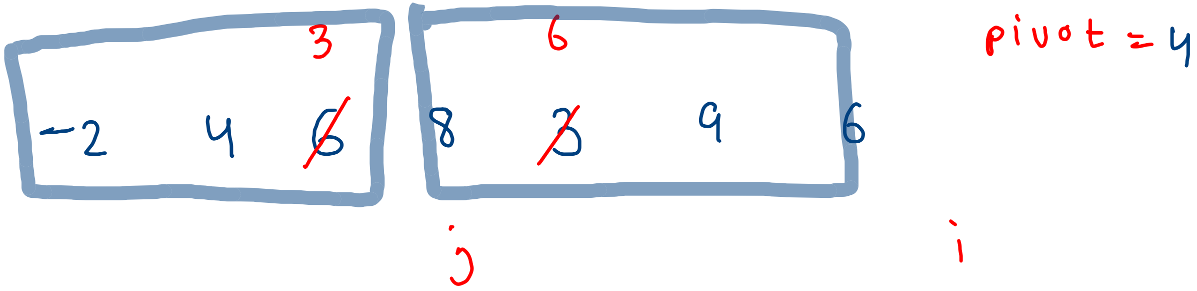
3 ~~8~~ 8

-2 4 6 ~~8~~ ~~3~~ 9 ~~6~~

j i



```
int i = 0;
int j = 0;

while(i < arr.length) {
    if(arr[i] > pivot) {
        i++;
    }
    else {
        swap(arr,i,j);
        i++;
        j++;
    }
}
```



# Quick Sort

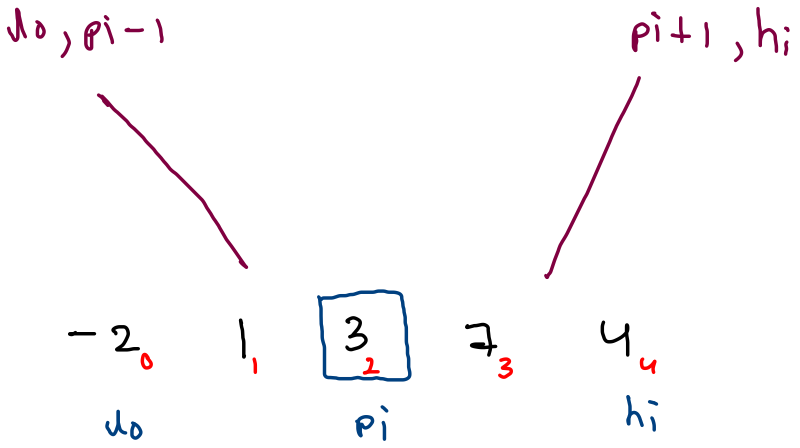
(i) sort

(ii) recursion

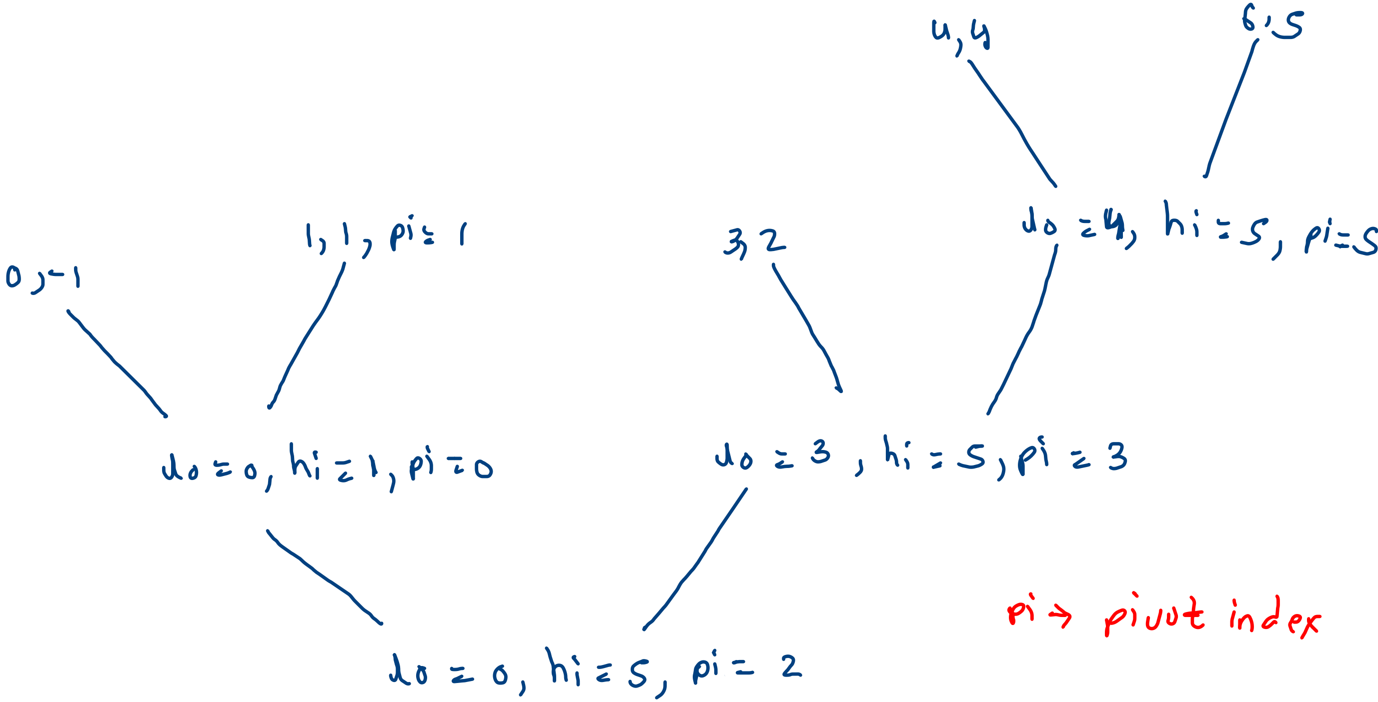
(iii) smaller problems:

```
qs (lo, pi-1);
qs (pi+1, hi);
```

7
-2
4
1
3



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| -2 | 3 | 3 | 4 | 6 | 9 |
| 0 | 1 | 2 | 3 | 4 | 5 |

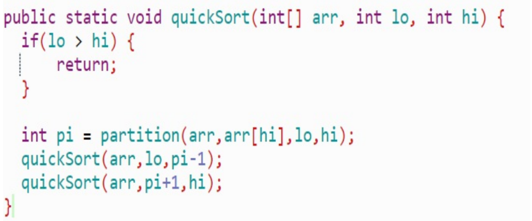


pi → pivot index

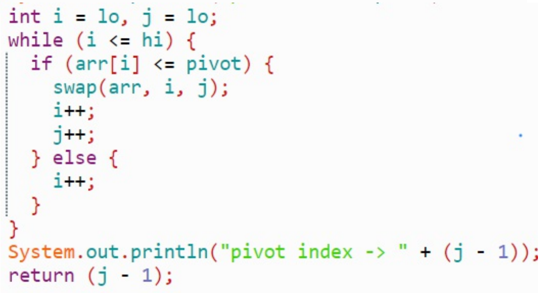
|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 6 | 3 | 4 | 9 | -2 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 |

| -2 | 3 | 3 | 4 | 6 | 9 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

pivot = 9

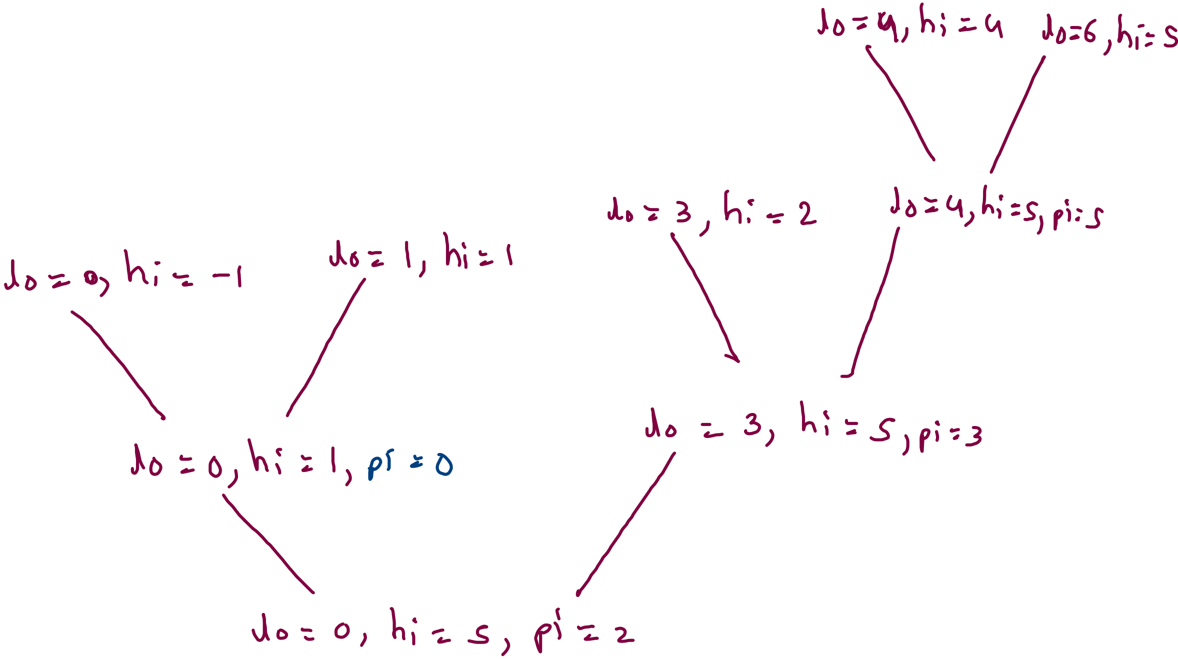


```
public static void quickSort(int[] arr, int lo, int hi) {
  if(lo > hi) {
      return;
  }

  int pi = partition(arr,arr[hi],lo,hi);
  quickSort(arr,lo,pi-1);
  quickSort(arr,pi+1,hi);
}
```



```
int i = lo, j = lo;
while (i <= hi) {
  if (arr[i] <= pivot) {
    swap(arr, i, j);
    i++;
    j++;
  } else {
    i++;
  }
}
System.out.println("pivot index -> " + (j - 1));
return (j - 1);
```

| 6 | 3 | 4 | 9 | -2 | 3 |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |



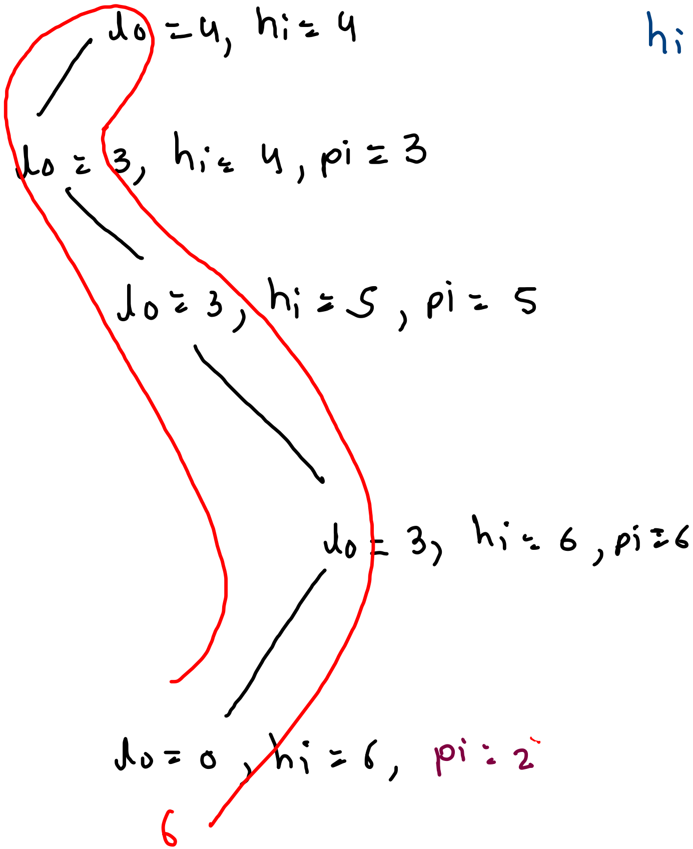
# Quick Select

| arr : | 2 | -1 | 4 | 5 | 6 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

lo hi

k = 5

sap = 4



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

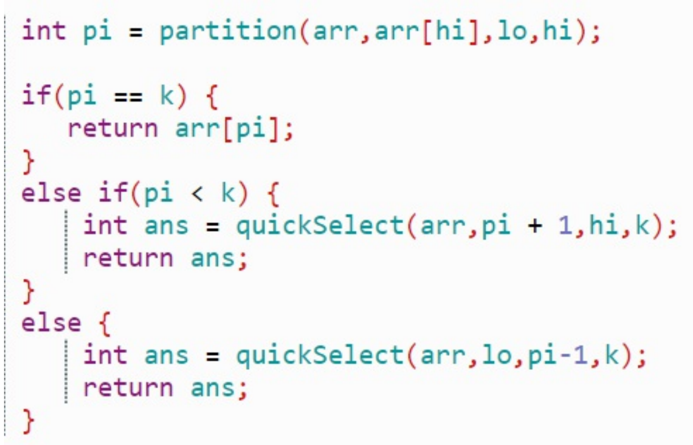
pivot = 5

k = 4

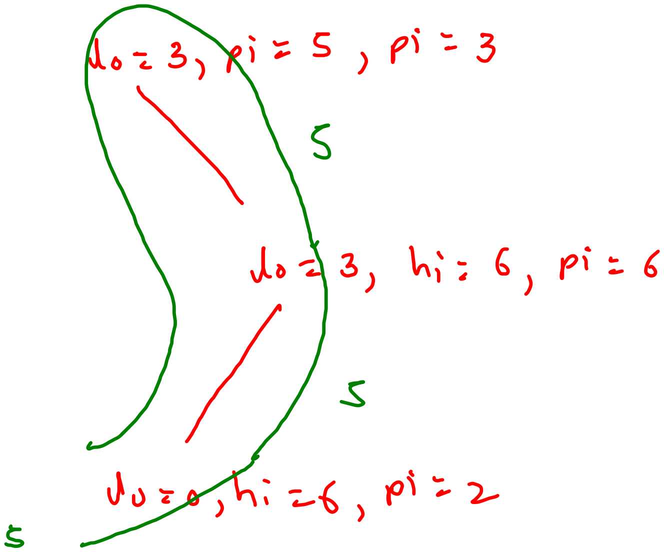
4th smallest

8ap -> 3

k = 3



```
int pi = partition(arr,arr[hi],lo,hi);

if(pi == k) {
   return arr[pi];
}
else if(pi < k) {
    int ans = quickSelect(arr,pi + 1,hi,k);
    return ans;
}
else {
    int ans = quickSelect(arr,lo,pi-1,k);
    return ans;
}
```



lo = 3, pi = 5 , pi = 3

5

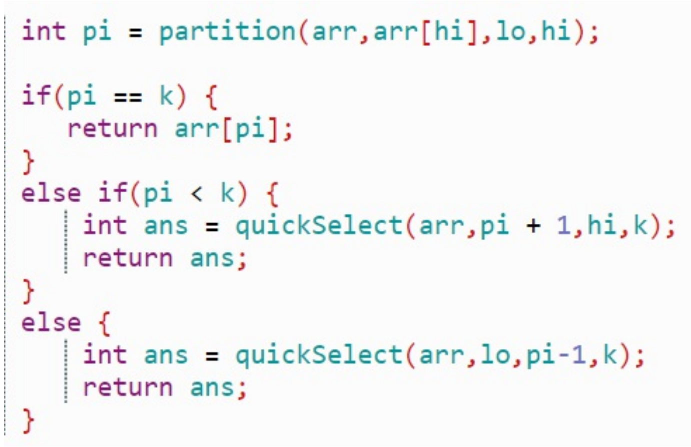
lo = 3, hi = 6, pi = 6

5

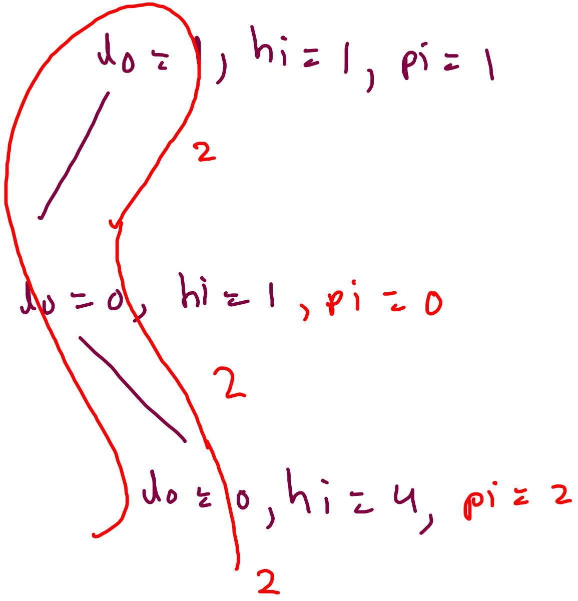
lo = 0, hi = 6, pi = 2

5

$-1_0$ $2_1$ $3_2$ $5_3$ $7_4$



```
int pi = partition(arr,arr[hi],lo,hi);

if(pi == k) {
   return arr[pi];
}
else if(pi < k) {
    int ans = quickSelect(arr,pi + 1,hi,k);
    return ans;
}
else {
    int ans = quickSelect(arr,lo,pi-1,k);
    return ans;
}
```



k = 2

$\boxed{\frac{sap = 1}{k = 1}}$