Timing Assignment

Analysis

Worst Case

```
1  int kthLargest (int* array,  int N, int k)
2  {
3      int low = 0;              // O(1)
4      int high = N-1;           // O(1)
5
6      while( low < high )       // cond: O(1) #: N-1
7      {
8        int start = low+1;      // O(1)
9        int stop = high;        // O(1)
10       while (start < stop)
11       {
12         if (array[start] > array[low])  // cond: O(1)
13           ++start;            // O(1)
14         else if (array[stop] <= array[low]) // cond: O(1)
15           --stop;             // O(1)
16         else
17         {
18           std::swap (array[start], array[stop]); // O(1)
19         }
20       }
21       if (array[start] < array[low])             // cond: O(1)
22         --start;              // O(1)
23       std::swap (array[low], array[start]);      // O(1)
24       if (start == k)
25         low = high = k;       // O(1)
26       else if (start > k)
27       {
28         high = start - 1;     // O(1)
29       }
30       else
31       {
32         low = start+1;        // O(1)
33       }
34     }
35     return array[low];        // O(1)
```

```
36 }
```

Evaluate condition block from lines 14-19 and then 12-19:

$$\begin{aligned} t_{if} &= O(1) + max(O(1), O(1)) \\ &= O(1 + 1) \\ &= O(1) \end{aligned}$$

```
1          ...
2          if (array[start] > array[low])  // cond: O(1)
3            ++start;             // O(1)
4          else if (array[stop] <= array[low]) // cond: O(1) total: O(1)
5            --stop;              // O(1)
6          else
7          {
8            std::swap (array[start], array[stop]); // O(1)
9          }
10         ...
```

Collapse

```
1         ...
2         if (array[start] > array[low])  // cond: O(1)
3           ++start;             // O(1)
4         else
5           // O(1)
6         ...
```

Evaluate

$$\begin{aligned} t_{if} &= O(1) + max(O(1), O(1)) \\ &= O(1 + 1) \\ &= O(1) \end{aligned}$$

Collapse

```
1  int kthLargest (int* array,  int N, int k)
2  {
3      int low = 0;               // O(1)
4      int high = N-1;            // O(1)
5
6      while( low < high )
```

```
7     {
8       int start = low+1;      // init: O(1)
9       int stop = high;        // init: O(1)
10      while (start < stop)    // cond: O(1) body: O(1) #: distance(start,
stop)
11      {
12        if (array[start] > array[low])      // cond: O(1) total: O(1)
13      }
14      if (array[start] < array[low])        // cond: O(1)
15        --start;              // O(1)
16      std::swap (array[low], array[start]); // O(1)
17      if (start == k)
18        low = high = k;       // O(1)
19      else if (start > k)
20      {
21        high = start - 1;     // O(1)
22      }
23      else
24      {
25        low = start+1;        // O(1)
26      }
27    }
28    return array[low];        // O(1)
29  }
```

Evaluate `while` loop from lines 10-13

$$
\begin{aligned}
t_{while} &= O(1) + \sum_{i=start}^{stop} (O(1) + O(1) + O(1)) \\
&= O(1) + O(distance(start, stop) * (O(1) + O(1) + O(1)) \\
&= O(1) + O(distance(start, stop)) \\
&= O(distance(start, stop)
\end{aligned}
$$

```
1   int kthLargest (int* array,  int N, int k)
2   {
3       int low = 0;              // O(1)
4       int high = N-1;           // O(1)
5
6       while( low < high )
7       {
8         int start = low+1;      // init: O(1)
9         int stop = high;        // init: O(1)
10        while (start < stop)    // cond: O(1) body: O(1) #: distance(start,
stop) total: O(distance(start, stop)
```

```
11
12        if (array[start] < array[low])        // cond: O(1)
13          --start;                // O(1)
14        std::swap (array[low], array[start]); // O(1)
15        if (start == k)
16          low = high = k;         // O(1)
17        else if (start > k)
18        {
19          high = start - 1;       // O(1)
20        }
21        else
22        {
23          low = start+1;          // O(1)
24        }
25      }
26      return array[low];          // O(1)
```

Collapsing loop:

```
1   int kthLargest (int* array,  int N, int k)
2   {
3       int low = 0;                // O(1)
4       int high = N-1;             // O(1)
5
6       while( low < high )
7       {
8         int start = low+1;        // init: O(1)
9         int stop = high;          // init: O(1)
10        // O(distance(start, stop)
11
12        if (array[start] < array[low])        // cond: O(1)
13          --start;                // O(1)
14        std::swap (array[low], array[start]); // O(1)
15        if (start == k)
16          low = high = k;         // O(1)
17        else if (start > k)
18        {
19          high = start - 1;       // O(1)
20        }
21        else
22        {
23          low = start+1;          // O(1)
24        }
25      }
26      return array[low];          // O(1)
```

Evaluate condition block from lines 12-13:

$$\begin{aligned} t_{if} &= O(1) + max(O(1), O(1)) \\ &= O(1+1) \\ &= O(1) \end{aligned}$$

And collapsing

```
1  int kthLargest (int* array,  int N, int k)
2  {
3      int low = 0;               // O(1)
4      int high = N-1;            // O(1)
5
6      while( low < high )
7      {
8        int start = low+1;       // init: O(1)
9        int stop = high;         // init: O(1)
10       // O(distance(start, stop)
11
12       if (array[start] < array[low])        // cond: O(1) total: O(1)
13
14       std::swap (array[low], array[start]); // O(1)
15       if (start == k)
16         low = high = k;        // O(1)
17       else if (start > k)      // cond: O(1)
18       {
19         high = start - 1;      // O(1)
20       }
21       else
22       {
23         low = start+1;         // O(1)
24       }
25     }
26     return array[low];         // O(1)
```

Evaluate condition block from lines 17-24 and then 15-24:

$$\begin{aligned} t_{if} &= O(1) + max(O(1), O(1)) \\ &= O(1) + O(1) \\ &= O(1) \end{aligned}$$

```
...
      if (start == k)
        low = high = k;        // O(1)
      else if (start > k)      // cond: O(1) total: O(1)
      {
        high = start - 1;      // O(1)
      }
      else
      {
        low = start+1;         // O(1)
      }
...
```

Collapse 17-24

```
...
      if (start == k)
        low = high = k;        // O(1)
      else
        // O(1)
...
```

Evaluate

$$\begin{aligned} t_{if} &= O(1) + max(O(1), O(1)) \\ &= O(1) + O(1) \\ &= O(1) \end{aligned}$$

Collapse

```
int kthLargest (int* array,  int N, int k)
{
    int low = 0;               // O(1)
    int high = N-1;            // O(1)

    while( low < high )
    {
      int start = low+1;       // init: O(1)
      int stop = high;         // init: O(1)
      // O(distance(start, stop)

      if (array[start] < array[low])        // total: O(1)
        --start;               // O(1)
```

```
14        std::swap (array[low], array[start]); // O(1)
15        if (start == k)          // total: O(1)
16      }
17      return array[low];         // O(1)
```

Sum `while` body and replace start/stop on line 10 with low+1 and high respectively.

$$\begin{aligned} t_{body} &= O(1+1+distance(low+1, high)+1+1+1) \\ &= O(distance(low, high)) \end{aligned}$$

```
1  int kthLargest (int* array,  int N, int k)
2  {
3      int low = 0;               // init: O(1)
4      int high = N-1;            // init: O(1)
5      while( low < high )
6      // body: O(distance(low, high))
7      return array[low];         // O(1)
```

Evaluate `while` loop

$$\begin{aligned} t_{while} &= O(1) + O(1) + \sum_{i=0}^{N-1}(O(distance(0, N-1))) \\ &= O(1+1+N^2) \\ &= O(N^2) \end{aligned}$$

```
1  int kthLargest (int* array,  int N, int k)
2    total: O(N^2)
```

Worst complexity of kthLargest = O(N^2)

Average Case