**Question 1. Design and Analysis of the algorithms**

1. (a) Consider an array of “Wooden blocks toys”. One of the features of these toys is that they are painted either Blue or Red. Devise an algorithm to keep all the Blue toys together at one end of the array and all Red toys together at the other end of the array. Is your algorithm in place? If not, what is the space complexity? What is the time complexity?

**Solution:**

For this problem, We use two pointers techniques as following:

* We initialize the left pointer at the beginning of array (represent for blue side)
* We initialize the right pointer at the end of array (represent for red side)
* We have the current element pointer (i), start from 0
* if the current element is blue, swap it with the element at the left pointer and move both left and i pointer to right
* if the current element is red, swap it with the element at the right pointer and move both right and I pointer to left
* if the current element is the correct position, just move i to right pointer

**Is your algorithm in place?**

Yes

**Space complexity:** O(1)

**Time Complexity:** O(n)

1. (b) Solve it for three different colors: Blue, Red and Green. . Is your algorithm in place? If not, what is the space complexity? What is the time complexity? Remember we are more concerned about the time complexity.

**Solution:**

For this problem, We use three pointers techniques as following:

* We initialize the left pointer at the beginning of array (represent for blue side)
* We initialize the middle pointer that start from 0 to traverse the array (represent for green side)
* We initialize the right pointer at the end of array (represent for red side)
* We traverse array with mid pointer
* if the current element is blue, swap it with the element at the left pointer and move both left and middle pointer to right
* if the current element is green, just move middle pointer to the right
* if the current element is red, swap it with the element at the right pointer and move right pointer to the left

**Is your algorithm in place?**

Yes

**Space complexity:** O(1)

**Time Complexity:** O(n)

(c) Solve it for four different colors: Blue, Red, Green and Yellow. Is your algorithm in place? If not, what is the space complexity? What is the time complexity? Remember we are more concerned about the time complexity.

**Solution:**

For this problem, We use four pointers techniques as following:

* We initialize the blue pointer at the beginning of array (represent for blue side)
* We initialize the red pointer to traverse the array
* We initialize the green pointer to keep track the boundary between green and yellow
* We initialize the yellow pointer at the end of array
* We traverse array with the red pointer
* If the current element is blue, swap it with the element at the blue pointer and move both the blue pointer and red pointer to the right
* If the current element is red, just move the red pointer to the right
* If the current element is green, swap it with the element at the green pointer and move the green pointer to the left
* If the current element is yellow, swap it with the element at the yellow pointer and move the yellow pointer to the left

**Is your algorithm in place?**

Yes

**Space complexity:** O(1)

**Time Complexity:** O(n)

**Question 2. Illustrate Quick sort. Since we do not have a random number generator, please pick a pivot so that they lead to alternating between “Good Self Call” and “Bad Self Call”.**

1. {1, 2, 3, 4, 5, 6, 7, 8, 9}
2. - Original array - Pivot 5 (Good Self Call) – Left {1, 2, 3, 4} – Right {6, 7, 8, 9}
3. - Left subarray {1, 2, 3, 4 } – Pivot 4 (Bad Self Call) – Left { 1, 2, 3} – Right {}
4. - Right subarray {6, 7, 8, 9} – Pivot 8 (Good Self Call) – Left {6, 7} – Right {9}
5. …
6. {8, 7, 6, 5, 4, 3, 2, 1, 9}
7. - Original array - Pivot 5 (Good Self Call) – Left {4, 3, 2, 1} – Right {8, 7, 6, 9}
8. - Left subarray {4, 3, 2, 1 } – Pivot 4 (Bad Self Call) – Left {} – Right {3, 2, 1}
9. - Right subarray {8, 7, 6, 9} – Pivot 8 (Good Self Call) – Left {7, 6} – Right {9}

…

1. {9, 1, 8, 2, 7, 3, 6, 4, 5}
2. - Original array - Pivot 5 (Good Self Call) – Left {1, 2, 3, 4} – Right {9, 8, 7, 6}
3. - Left subarray {1, 2, 3, 4 } – Pivot 4 (Bad Self Call) – Left {1, 2, 3} – Right {}
4. - Right subarray {9, 8, 7, 6} – Pivot 8 (Good Self Call) – Left {7, 6} – Right {9}
5. …
6. {5, 1, 4, 2, 3, 9, 7, 6, 8}
7. - Original array - Pivot 5 (Good Self Call) – Left {1, 4, 2, 3} – Right {9, 7, 6, 8}
8. - Left subarray {1, 4, 2, 3} – Pivot 4 (Bad Self Call) – Left {1, 2, 3} – Right {}
9. - Right subarray {9, 8, 7, 6} – Pivot 8 (Good Self Call) – Left {7, 6} – Right {9}
10. …

**Question 3. Illustrate quickSelect. Since we do not have a random number generator, please pick a pivot so that they lead to alternating between “Good Self Call” and “Bad Self Call”.**

1. {1, 2, 3, 4, 5, 6, 7, 8, 9} k = 5
2. - Original array - Pivot 5 (Good Self Call) – Left {1, 4, 2, 3} – Right {6, 7, 8, 9} -> 5th smallest = 5
3. {8, 7, 6, 5, 4, 3, 2, 1, 9} k = 3
4. - Original array - Pivot 5 (Good Self Call) – Left {4, 3, 2, 1} – Right {8, 7, 6, 9}
5. - Left subarray {4, 3, 2, 1} – Pivot 4 (Bad Self Call) – Left {3, 2, 1} – Right {}
6. - Left subarray {3, 2, 1} – Pivot 3 (Bad Self Call) – Left {2, 1} – Right {} -> 3rd samllest = 3
7. …
8. {9, 1, 8, 2, 7, 3, 6, 4, 5} k = 8
9. - Original array - Pivot 5 (Good Self Call) – Left {1, 2, 3, 4} – Right {9, 8, 7, 6}
10. - Right subarray {9, 8, 7, 6} – Pivot 8 (Good Self Call) – Left {7, 6} – Right {9}
11. - Right subarray {9} – Pivot 9 (Bad Self Call) – Left {} – Right {} -> 8th smallest = 9
12. …
13. {5, 1, 4, 2, 3, 9, 7, 6, 8} k = 5
14. - Original array - Pivot 5 (Good Self Call) – Left {1, 4, 2, 3} – Right {9, 7, 6, 8} -> 5th smallest = 5

**Question 4. Exploration**

**Let us redefine “Good Self Call” and “Bad Self Call”**

**Good self-call: the sizes of *L* and *G* are each less than 2*n*/3 (normal division)**

**Bad self-call: one of *L* and *G* has size greater than or equal to 2*n*/3.**

1. (a) Repeat the calculations shown in Slides 15, 16 and 17. (You need not draw pictures).
2. -> It’s the same result in Slides 15
3. (b) Are you able to derive the same results in Slides 16 and 17? If not, why?
4. -> It’s the same result in Slides 16 and 17