1. พิจารณาโปรแกรม Quick Select ด้านล่าง สำหรับค้นหาสมาชิกในอาร์เรย์ arr ที่มีค่าน้อยที่สุดลำดับที่ 4 (k=4)

```
#include<stdio.h>
int arr[] = {1, 5, 10, 4, 8, 2, 6, 9, 20};
int k=4;
int n = sizeof(arr) / sizeof(arr[0]);

int partition(int l, int r) {
       // **** use median of three for pivot selection

}
int quickSelect(int low, int high, int k) {

    if(low == high)
       return ..........;

 int p = partition(arr, low, high);

    if (.....................)        // case k = Pivot position
        return ..............;
    else if (.................)                // case k ∈ L
        return quickSelect(arr, low, p-1, k);
    else {                                      // case k ∈ R
           k = ...................;
           return quickSelect(arr, ……… , ……… , …… );
         }
}

int main() {
    printf("%d", quickSelect(arr, 0, n - 1, k));
    return 0;
}
```

1.1 จงเติมโค้ดด้านบนเพื่อให้โปรแกรมทำงานได้อย่างสมบูรณ์

1.2 แสดงผลลัพธ์ของการทำ partition ในแต่ละ step

1.3 จงวิเคราะห์เวลาทำงานของอัลกอริทึมนี้

2. กำหนดให้ X = 342 และ Y = 231 แสดงผลลัพธ์ในแต่ละขั้นตอนการหาผลคูณ X*Y ด้วยอัลกอริทึม Karasuba

3. กำหนดให้ P = { (7, 2), (3, 1), (9, 3), (4, 5), (1, 4), (6, 9), (2, 6), (5, 7), (8, 6) } จงวาด recursive tree เพื่อแสดงขั้นตอนการค้นหา maxima set ด้วยวิธี divide and conquer พร้อมหาจำนวนครั้งในการเปรียบเทียบสมาชิกใน M1 และ M2 เพื่อรวมคำตอบ

4. พิจารณา Pseudo code ด้านล่าง เพื่อหาสมาชิกในอาร์เรย์ A ที่มีค่าใกล้เคียงกับ M มากที่สุด K จำนวน

```
Algorithm Search(A, M, k)
    Input: Array A, target M and k: number of nearest items
    Output: A[left] .... A[right]
1. Sort A in ascending order
2. Search the i^th index in A for M
3. left = i-1, right = i
4. while (right - left) <= k:
5.      print left, right
6.       if abs(A[left]- M) > abs(A[right]- M)
7.          right = right + 1
8.       else
9.          left = left - 1
10. end algorithm
```

4.1 จงแสดงขั้นตอนการทำงานของอัลกอริทึม โดยใช้ข้อมูล A[] = {10, 12, 15, 17, 18, 20, 25} เมื่อ k = 2 และ M = 8

4.2 จงวิเคราะห์ time complexity ของอัลกอริทึมนี้ (ไม่รวมเวลา sorting **)

4.3 จงพัฒนาอัลกอริทึมดังกล่าว โดยแสดงค่าของ left และ right โดยใช้อินพุตตามข้อ 4.1

4.4 จงปรับปรุงอัลกอริทึมในข้อ 4.3 โดยใช้หลักการ divide and conquer (**O(log n)**)