# Answer to Module 6 Problem Set

- Student Name: Gaole Dai
- Student ID: S01435587

## Problem 1

The 7th smallest element is **9**.

Python code for quickselect algorithm:

```
def qspart(a, lo, hi):
    if hi == lo:
        return lo
    i = lo + 1
    j = hi
    pivot = a[lo]
    while(True):
        while(a[i] < pivot):
            i += 1
            if i >= hi:
                break
        while (pivot < a[j]):
            j -= 1
            if (j == lo):
                break
        if i >= j:
            break
        a[j], a[i] = a[i], a[j]
    a[j], a[lo] = a[lo], a[j]

    return j

def qselect(a, k):
    lo, hi = 0, len(a) - 1
    while lo <= hi:
        print(f'before part({a[lo:hi+1]})')
        m = qspart(a, lo, hi)
        print(f'after part({a[lo:hi+1]})')
        if (m == k):
            print(f'{a[m]}')
            return a[m]
        if m < k:
            lo = m+1
        else:
            hi = m-1
```

```
aa = [3, 17, -5, 4, 13, 8, 7, 6, 9]
qselect(t := aa[:], 6)
```

The index 6 element (7th smallest element) is **9**, where the trace is:

```
before part([3, 17, -5, 4, 13, 8, 7, 6, 9])
after part([-5, 3, 17, 4, 13, 8, 7, 6, 9])
before part([17, 4, 13, 8, 7, 6, 9])
after part([9, 4, 13, 8, 7, 6, 17])
before part([9, 4, 13, 8, 7, 6])
after part([7, 4, 6, 8, 9, 13])
9
```

## Problem 2

The median is **6**.

```
bb = [9, 8, 6, 4, -100]
qselect(t := bb[:], len(bb) // 2)
```

Using the algorithm shown in Problem 1, the median is **6**, and the trace is:

```
before part([9, 8, 6, 4, -100])
after part([-100, 8, 6, 4, 9])
before part([-100, 8, 6, 4])
after part([-100, 8, 6, 4])
before part([8, 6, 4])
after part([4, 6, 8])
before part([4, 6])
after part([4, 6])
before part([6])
after part([6])
6
```

## Problem 3

The possible pivot elements are **4, 5 and 9**.

According to the quickselect algorithm, the pivot should be smaller than the right side elements and bigger than the left side elements.

```
3   1   2   4   5   8   7   6   9
```

- 3 is larger than right elements such as 1 2 ...
- 1 is smaller than the left element 3
- 2 is smaller than the left element 3
- 4 fits (elements on the left are smaller than 4, and element on the right are larger)
- 5 fits (elements on the left are smaller than 5, and element on the right are larger)
- 8 is larger than right element 7 and 9
- 6 is smaller than left element 7 and 8
- 9 fits (all the element on left is smaller than 9)

## Problem 4

The random number sequence is **[0, 1, 1, 1, 3]**.

Shuffle algorithm:

```
import random as rn

def kshuffle(a):
    for i in range(len(a)):
        r = rn.randint(0, i)
        a[i], a[r] = a[r], a[i]
    return a
```

Use the backwards track, and we could obtain the random number sequence [0, 1, 1, 1, 3].

| Input | idx | random number | Output |
|---|---|---|---|
| 1 4 2 3 5 | 4 | 3 | 1 4 2 5 3 |
| 1 3 2 4 5 | 3 | 1 | 1 4 2 3 5 |
| 1 2 3 4 5 | 2 | 1 | 1 3 2 4 5 |
| 1 2 3 4 5 | 1 | 1 | 1 2 3 4 5 |
| 1 2 3 4 5 | 0 | 0 | 1 2 3 4 5 |

## Problem 5

**a**. The total number of permutations $N^N$. Since the outer loop runs $N$ times, and the possibilities for random number is $N$, the total number of permutations is $N^N$.

```
import random as rn

def bshuffle(a):
    for i in range(len(a)):
        r = rn.randint(0, len(a) - 1)
        a[i], a[r] = a[r], a[i]
    return a
```

**b**. The total number of permutations generated by KFY shuffling algorithm is $N!$. Since the outer loop runs $N$ times, and the possibilities for random number is $i$, the total number of permutations is then $1 \times 2 \times 3 \times 4 \times \ldots \times N$, which is $N!$.

```
import random as rn

def kshuffle(a):
    for i in range(len(a)):
        r = rn.randint(0, i)
        a[i], a[r] = a[r], a[i]
    return a
```

**c**. The total number of permutations of $N$ things is $N!$, the number of $xseq\ bshuffle$ is $N^N$, and the number of $xseq\ KFY$ is $N!$. For $KFY$ shuffle, all permutations are of equal probability. However, for $bshuffle$, the possibility of permutations is not equally likely which result in the bias, because the `rand()` function produce integer among 0 to the length of array, an array element could be changed for many times, but for $KFY$ shuffle it would be changed once.

If $N = 3$, the number of permutation is 6, which is `123, 132, 213, 231, 312, 321.` With $bshuffule$, it would produce $3^3(27)$ possible outcomes, each of the output would map to the un-biased shuffle results as listed before.  Since 27 is not evenly divisible by six, there *must* be over-represented among the combinations.