VE477 Lab7

Q1

1. `RandomSearch`

   ```
   # random search array A for a given key k
   def RandomSearch(A, k):
       index_set = set()
       n = len(A)
       while len(index_set) < n:
           index = random.randint(0, n - 1)
           if A[index] == k:
               return index
           index_set.add(index)
       return n  # not found, index out of range
   ```

2. a) **no index**: average number of picked index is approximately $n \ln n$

   Define each time we found a new index as a *hit*, so we want to know the expected number x to obtain n *hits*. Partition x into stages, such that the i th stage contains the choosing between $i-1$th hit and ith hit.

   For each chosen during the ith stage, $(i-1)$ index have been chosen, $(n-i+1)$ index have not been chosen. So the probability of getting a hit is $(n-i+1)/n$.

   Use x_i to denote the number of chosen during stage i. The total number of chosen is $x = \sum_{i=1}^{n} x_i$. $E[x_i] = n/(n-i+1)$. Then we get

   $$E[x] = E[\sum_{i=1}^{n} x_i] = \sum_{i=1}^{n} E[x_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n\sum_{i=1}^{n} \frac{1}{i} = n(\ln n + O(1))$$

   b) **one index**: average number is n.

Then each pick is a bernoulli trial with parameter $p = 1/n$, to get a total number of success as 1, from the expectation of binomial distribution, we should pick n indices.

c) **more than one index (assume m)**: average number is n/m.

Same as b), parameter $p = m/n$, so from expectation should be n/m.

Q2

1. `LinearSearch`

```
def LinearSearch(A, k):
    n = len(A)
    for i in range(n):
        if A[i] == k:
            return i
    return n  # not found, index out of range
```

2. a) **no index**: average and worse case should both be n

b) **one index**: average $\frac{(n+1)}{2}$, worse n

Average case: examine the expected running time, assume all permutation are equally like to happen, then the probability that k occur on each index is the same as $1/n$.

$E = 1 \cdot \frac{1}{n} + 1 \cdot \frac{2}{n} + \ldots + 1 \cdot \frac{n}{n} = \frac{(n+1)}{2}$

c) **more than one index (assume m)**: average $\frac{n}{m}$, worse $n - m + 1$

Assume all the permutations occur equally likely. Let i be the number of index picked to find the key.
$Pr[i = 1] = \frac{m}{n}, Pr[i = 2] = \frac{n-m}{n} \cdot \frac{m}{n-1} \ldots Pr[i] < \frac{m}{n} \cdot (\frac{n-m}{n-1})^{i-1}$.

$$E[X] = \sum_{i=1}^{n-m+1} i \cdot Pr[i] < \frac{n}{m}$$

Q3

1. `ScrambleSearch`

   ```
   def ScrambleSearch(A, k):
       random.shuffle(A)
       return LinearSearch(A, k)
   ```

2. same as Linear Search

Q4

`ScrambleSearch` is the best.

Based on the analysis, as `LinearSearch` and `ScrambleSearch` have the same time but `LinearSearch` need the equally likely assumption.

Q5

```
1000 times random search time: 28.78109426004812
1000 times linear search time: 0.0993400810608268
1000 times linear search time: 1.3530246154405177
```

1. `LinearSearch` is the best in practice.
2. Practice give different answer to the theoretical answer, because in reality, the random permutation time is much longer.

```
if __name__ == '__main__':
    random.seed()
    arr_len = 1000000
    arr = [random.randint(0, arr_len * 10) for i in range(arr_len)]
    random_time, linear_time, scramble_time = [], [], []
    for i in range(1000):
        key = random.randint(0, arr_len)
        random_time.append(timeit.timeit("RandomSearch(arr, key)",
                                         setup="from __main__
import RandomSearch, arr, key", number=3))
        # print("1000 times random search time:
{}".format(random_time))
        linear_time.append(timeit.timeit("LinearSearch(arr, key)",
```

```
                                                    setup="from __main__
import LinearSearch, arr, key", number=3))
        # print("1000 times linear search time:
{}".format(linear_time))
        scramble_time.append(timeit.timeit("ScrambleSearch(arr,
key)",
                                                      setup="from __main__
import ScrambleSearch, arr, key", number=3))
        # print("1000 times linear search time:
{}".format(scramble_time))
    print("1000 times random search time:
{}".format(np.mean(random_time)))
    print("1000 times linear search time:
{}".format(np.mean(linear_time)))
    print("1000 times linear search time:
{}".format(np.mean(scramble_time)))
```