Algorithms and Lab

한국기술교육대학교
KOREATECH

	Lab 05
ID	2020136149
Name	김태섭

2023. 04. 12

1-1. InterpolationSearch

```
def interpolationSearchRec(self,left, right, x, A):

    if (left<= right and x >= A[left] and x <= A[right]):
        mid = left + ((x - A[left]) * (right - left)) // (A[right] - A[left])
        if x == A[mid]:
            return mid
        elif x < A[mid]:
            return self.interpolationSearchRec(left, mid-1, x, A)
        else:
            return self.interpolationSearchRec(mid+1, right, x, A)
    return -1
```

```
def interpolationSearchItr(self,A, x):
    (left, right) = (0, len(A) - 1)

    while A[right] != A[left] and A[left] <= x <= A[right]:
        mid = left + (x - A[left]) * (right - left) // (A[right] - A[left])
        if x == A[mid]:
            return mid
        elif x < A[mid]:
            right = mid - 1
        else:
            left = mid + 1
    return -1
```

```
def ternarySearchTest():
    lab05 = Lab05()
    A = random.sample(range(1, 50), 30)
    A.sort()
    print(A)
    key = 19
    #p = lab05.ternarySearchRec(0, len(A) - 1, key, A)
    #print("ternarySearchRec: Index of", key, "is", p)
    #p = lab05.ternarySearchItr(A, key)
    #print("ternarySearchItr: Index of", key, "is", p)
    p = lab05.interpolationSearchRec(0, len(A) - 1, key, A)
    print("interpolationSearchRec: Index of", key, "is", p)
    p = lab05.interpolationSearchItr(A, key)
    print("interpolationSearchItr: Index of", key, "is", p)
```

Output:

```
r"  64494"  --  "C:\Users\kyung\OneDrive\Desktop\컴프\main5.py"
[2, 6, 7, 8, 9, 10, 15, 16, 17, 19, 22, 23, 25, 26, 28, 29, 30, 32, 35, 36, 38,
39, 40, 42, 43, 44, 46, 47, 48, 49]
interpolationSearchRec: Index of 19 is 9
interpolationSearchItr: Index of 19 is 9
```

Code description: interpolation search is an algorithm that searches by reducing the range of sorted lists. Create list A by randomly extracting 30 numbers from 1 to 50. Next, use the sort() function to sort, save the number you want to find in the key, and put the key in the interpolation search function to tell you what number key you want to find in the list. However, if there is no number, return -1.

complexity Function:

$$\ominus(\log(\log n))$$

1-2. findSL

- Search Largest

```
def findSL(self, S):
    n = len(S)
    small = large = S[0]
    for i in range(1, n):
        if (S[i] > large):
            large = S[i]


        if (small > S[i]):
            small = S[i]
    return large
```

- Search Smallest

```
def findSL(self, S):
    n = len(S)
    small = large = S[0]
    for i in range(1, n):
        if (S[i] > large):
            large = S[i]


        if (small > S[i]):
            small = S[i]
    return small
```

```
def findSLTest():
    lab05 = Lab05()
    A = random.sample(range(1,50), 30)
    A.sort()
    print(A)
    print('findSL result is: ', lab05.findSL(A))
```

Output:

- Search Largest

```
[2, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 18, 19, 21, 23, 24, 25,
26, 27, 29, 30, 31, 33, 39, 40, 41, 43, 46, 47, 49]
find Largest result is:  49
```

- Search Smallest

```
[3, 8, 9, 10, 13, 15, 16, 17, 19, 20, 21, 23, 24, 27, 28, 29, 3
0, 31, 33, 34, 35, 36, 38, 41, 43, 44, 45, 47, 48, 49]
find Smallest result is:  3
```

complexity Function:

$$T(n) = 2(n-1)$$

1-3. findSLPK

- Search Largest

```
def findSLPK(self, S):
    n = len(S)
    if S[0] < S[1]:
        small = S[0]
        large = S[1]
    else:
        small = S[1]
        large = S[0]

    for i in range(2, n, 2):
        if (S[i] < S[i+1]):
            if (S[i] < small):
                small = S[i]
            if (S[i+1] > large):
                large = S[i+1]
        else:
            if (S[i+1] < small):
                small = S[i+1]
            if (S[i] > large):
                large = S[i]

    return large
```

- Search Smallest

```
def findSLPK(self, S):
    n = len(S)
    if S[0] < S[1]:
        small = S[0]
        large = S[1]
    else:
        small = S[1]
        large = S[0]

    for i in range(2, n, 2):
        if (S[i] < S[i+1]):
            if (S[i] < small):
                small = S[i]
            if (S[i+1] > large):
                large = S[i+1]
        else:
            if (S[i+1] < small):
                small = S[i+1]
            if (S[i] > large):
                large = S[i]

    return small
```

Output:

- Search Largest

```
[5, 6, 8, 9, 11, 12, 13, 15, 16, 18, 19, 20, 21, 22, 23, 25, 27
, 28, 29, 32, 33, 35, 36, 39, 40, 42, 43, 44, 45, 48]
findSLPK result is  48
```

- Search Smallest

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 17, 18, 19, 20, 21, 22,
 24, 26, 30, 31, 34, 37, 38, 41, 43, 44, 45, 49]
findSLPK result is  1
```

complexity Function:

$$T(n) = \begin{cases} \frac{3n}{2} - 2 & n \text{ is even} \\ \frac{3n}{2} - \frac{2}{3} & n \text{ is odd} \end{cases}$$

1-4. selectionRank

```
def rankSelection(self,A, low, high, k):
    pos = self.partition(A, low, high)

    if (pos+1 == low+k):
        return A[pos]
    elif (pos+1 > low+k):
        return self.rankSelection(A, low, pos-1, k)
    else :
        return self.rankSelection(A, pos+1, high, k-(pos+1-low))
```

```
def partition(self,A, low, high):
    pivot = A[low]
    j=low
    for i in range(low+1,high+1):
        if (A[i] < pivot):
            j+=1
            A[j],A[i]=A[i],A[j]
    A[j],A[low]=A[low],A[j]
    return j
```

```
def rankSelectionTest():
    lab05=Lab05()
    A = random.sample(range(1, 50), 30)
    print(A)
    medi=int(round((1+len(A))/2))
    largest= len(A)
    smallest=1
    print("element at rank {} is {} ".format(smallest, lab05.rankSelection(A, 0, len(A)-1, smallest)))
    print("element at rank {} is {} ".format(largest, lab05.rankSelection(A, 0, len(A) - 1, largest)))
    print("element at rank {} is {} ".format(medi, lab05.rankSelection(A, 0, len(A) - 1, medi)))
    A.sort()
    print(A)
```

Output:

```
[39, 3, 38, 25, 14, 41, 40, 20, 27, 31, 45, 9, 4, 22, 35, 19, 30
, 10, 28, 36, 8, 16, 15, 44, 11, 37, 34, 24, 49, 29]
element at rank 1 is 3
element at rank 30 is 49
element at rank 16 is 28
[3, 4, 8, 9, 10, 11, 14, 15, 16, 19, 20, 22, 24, 25, 27, 28, 29,
 30, 31, 34, 35, 36, 37, 38, 39, 40, 41, 44, 45, 49]
```

complexity Function:

$$T(n) = \sum_{i=2}^{n} 1 = n - 1$$

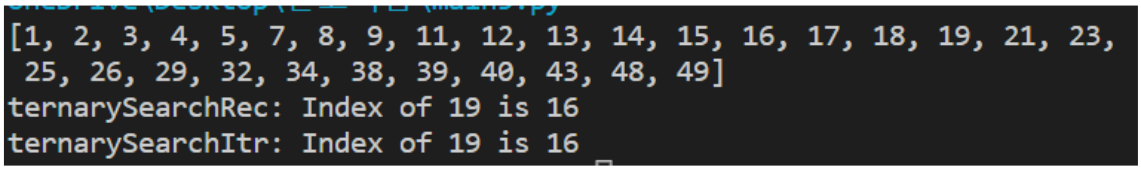
1-5. ternarySearch

```
def ternarySearchItr(self,A, x):
    (left, right) = (0, len(A) - 1)
    while left <= right:
        mid1 = left + (right - left) // 3
        mid2 = right - (right - left) // 3
        if A[mid1] == x:
            return mid1
        elif A[mid2] == x:
            return mid2
        elif A[mid1] > x:
            right = mid1 - 1
        elif A[mid2] < x:
            left = mid2 + 1
        else:
            left = mid1 + 1
            right = mid2 - 1

    return -1
```

```
def ternarySearchRec(self,left, right, x, A):
    if (right >= left):
        mid1 = left + (right - left) //3
        mid2 = right - (right - left) //3

        if (A[mid1] == x):
            return mid1
        if (A[mid2] == x):
            return mid2
        if (x < A[mid1]):
            return self.ternarySearchRec(left, mid1 - 1, x, A)
        elif (x > A[mid2]):
            return self.ternarySearchRec(mid2 + 1, right, x, A)
        else:
            return self.ternarySearchRec(mid1 + 1, mid2 - 1, x, A)
    return -1
```

```
def ternarySearchTest():
    lab05 = Lab05()
    A = random.sample(range(1, 50), 30)
    A.sort()
    print(A)
    key = 19
    p = lab05.ternarySearchRec(0, len(A) - 1, key, A)
    print("ternarySearchRec: Index of", key, "is", p)
    p = lab05.ternarySearchItr(A, key)
    print("ternarySearchItr: Index of", key, "is", p)
    #p = lab05.interpolationSearchRec(0, len(A) - 1, key, A)
    #print("interpolationSearchRec: Index of", key, "is", p)
    #p = lab05.interpolationSearchItr(A, key)
    #print("interpolationSearchItr: Index of", key, "is", p)
```

Output:



complexity Function:

$$T(n) = 4\log_3 n + O(1)$$

2-1. Post-office location problem

```
def postOfficeLocation(self, P):
    Xs, Ys = zip(*P)
    Xs=list(Xs)
    Ys=list(Ys)
    print (Xs)
    print (Ys)
    ax=sum(Xs)//len(Xs)
    ay=sum(Ys)//len(Ys)
    ap=(ax,ay)

    mx=self.rankSelection(Xs, 0, len(Xs)-1, len(Xs)//2)
    my=self.rankSelection(Ys, 0, len(Ys)-1, len(Ys)//2)
    mp=(mx,my)
    mind=0
    for i in range(len(P)):
        mind+=self.Ed(ap,P[i])
    print("total distance from (average point) ({},{}) using Euclidean  = {:.2f}".format(ax,ay,mind))

    mind=0
    for i in range(len(P)):
        mind+=self.Md(ap,P[i])
    print("total distance from (average point) ({},{}) using Manhattan = {:.2f}".format(ax,ay,mind))

    mind=0
    for i in range(len(P)):
        mind+=self.Ed(mp,P[i])
    print("total distance from (median point) ({},{}) using Euclidean = {:.2f}".format(mx,my,mind))

    mind=0
    for i in range(len(P)):
        mind+=self.Md(mp,P[i])
    print("total distance from (median point) ({},{}) using Manhattan = {:.2f}".format(mx,my,mind))
```

```
def POLTest():
    lab05 = Lab05()
    P = [(1, 1), (3, 5), (4, 6), (6, 6), (2, 3), (3, 5), (7, 3), (12, 3), (4, 14)]
    lab05.postOfficeLocation(P)
```

Output:

```
[1, 5, 6, 6, 3, 5, 3, 3, 14]
total distance from (average point) (4,5) using Euclidean  = 33.92
total distance from (average point) (4,5) using Manhattan = 41.00
total distance from (median point) (3,3) using Euclidean = 39.28
total distance from (median point) (3,3) using Manhattan = 44.00
```

complexity Function:

$$minDist = \frac{1}{n}\sum_{i=1}^{n} d(p, V_i)$$

2-2. Fake-coin problem

```
def __init__(self, n=10):
    self.n = n
    self.coins=self.random_coins(n)

def random_coins(self,n):
    coins = [2] * (n - 1) + [1] * (1)
    random.shuffle(coins)
    print(coins)
    return coins
```

```
def fake_coin_DQ2(self):
    pile = list(self.coins)
    while len(pile) > 1:
        n = len(pile) // 2
        A = pile[:n]
        B = pile[n : 2 * n]
        result = self.compare(A, B)
        if result == Scale.LeftHeavy:
            pile = B
        elif result == Scale.RightHeavy:
            pile = A
        elif result == Scale.Balanced:
            C = pile[2 * n :]
            pile = C
    return pile[0]
```

```
def fackCoinTest():
    fc = FakeCoin(20)
    print("Fake Coin with weight:", fc.fake_coin_DQ2())
```

Output:

```
[2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
Fake Coin with weight: 1
```

complexity Function:

$$T(n) = \begin{cases} T(1) = 1 & n = 1 \\ T(\lceil \frac{n}{3} \rceil) + 2 & n > 1 \end{cases}$$

2-3. Finding bit flips problem

```
def flip(self,ch):
    return '1' if (ch == '0') else '0'

def minBitFlips(self,start: int, goal: int) -> int:
    start = bin(start)[2:]
    goal = bin(goal)[2:]
    res = 0
    if len(start) < len(goal):
        start = "0" * (len(goal) - len(start)) + start
    elif len(start) > len(goal):
        goal = "0" * (len(start) - len(goal)) + goal

    print(" start {} , goal {}".format(start, goal))
    startc = list(start)
    for i in range(len(start)):

        if start[i] != goal[i]:
            ch=start[i]
            startc[i]=self.flip(ch)
            print(startc)
            res += 1
    return res
```

```
def bitflipstest():
    bf=BitFlips()
    res = bf.minBitFlips(3, 11193)
    print("No. of flips :",res)
```

Output:

```
 start 00000000000011 , goal 10101110111001
['1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '1', '1', '1', '0', '1', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1']
['1', '0', '1', '0', '1', '1', '1', '0', '1', '1', '1', '0', '1', '1']
['1', '0', '1', '0', '1', '1', '1', '0', '1', '1', '1', '0', '0', '1']
No. of flips : 9
```

complexity Function:

$$O(\log(\max(start, goal)))$$