1. Preliminaries

2.0: 본 과제는 Python 3.9 64bit환경에서 진행되었습니다. 폴더는 (input_20000size) (input_1000000size)가 있으며, 파일은 algorithm.py, main.py가 있습니다. main.py를 실행시키면 input.txt가 있을시 random.txt, deter.txt, result.txt를 만듭니다.

2.1 Random Selection

주어진 input에서 k번째 원소를 찾는 알고리즘으로, Quick Sort처럼 Partition 알고리즘을 이용해 input을 pivot기준으로 왼쪽은 pivot보다 작거나 같고, 오른쪽은 pivot보다 크게 분할한다. pivot이 답이면 pivot을 반환, pivot이 작으면 분할 된 sub-array중 왼쪽을, 크면 오른쪽에서 다시 Selection을 진행하는 알고리즘이다. 이때 Partition 할 때 무작위로 기준 원소를 선정해 최악의 경우가 반복되어 $O(n^2)$이 될 확률을 줄여주는 알고리즘이다. pseudo code는 다음과 같다. 1)

```
random_select(input, p, r, k){
  if p = r then return input[p];
  q = random_partition(input,p,r);
  i = q - p + 1;
  if k == i then return input[q];
  elif k > i then return random_select(input, p, q-1, k);
  else return random_select(input, q+1,r, k-i);
  end if
}
random_partition(input, p, r){
  a = randomint(p,r);
  input[r] ⇔ input[a];
  key = input[r];
  i = p - 1;
  for j = p to r-1
    if input[j] ≦ key then
        i++;
        input[i] ⇔ input[j];
    A[i+1] ⇔ A[r];
    end if
  end for
  return i+1;
}
```

위 알고리즘의 Time Complexity는 average의 경우 pivot이 전체에서 i번째로 작은 원소일때 다음과 같다.

$$T(n) \leqq \max(T(n-i), T(i-1)) + \Theta(n)$$

sub array가 각각 $n-i$, $i-1$개로 나뉘었으므로, $\max(T(n-i), T(i-1))$은 이 두개의 subarray 중 오래걸리는 쪽을 고르는 것이다.

$\Theta(n)$은 재귀로 호출하는 부분을 제외한 부분으로, Partition함수에서 대부분이 소요된다.

전체 배열에서 i가 $1 \sim n$까지 동등한 확률을 가질때

$$\leqq \frac{1}{n}\sum_{i=1}^{n}\max[T(i-1), T(n-i)] + \Theta(n)$$

$$\leqq \frac{2}{n}\sum_{i=\lfloor n/2 \rfloor}^{n-1} T(i) + \Theta(n)$$

귀납적 가정으로 $\lfloor n/2 \rfloor \leqq i \leqq n$인 모든 i에 대해 $T(i) \leqq ci$라 가정하자.

$$\leqq \frac{2}{n}\sum_{i=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$$

$$= \frac{2}{n}\left(\sum_{i=1}^{n-1} ci - \sum_{i=1}^{\lfloor n/2 \rfloor - 1} ci\right) + \Theta(n)$$

$$= cn + \left(-\frac{cn}{4} + \frac{c}{2} - \frac{2c}{n} + \Theta(n)\right) \leqq cn$$

즉 $T(n) = O(n)$, 전체원소를 한번 훑는데 최소 $\Omega(n)$가 필요하므로 $T(n) = \Theta(n)$

Worst Case의 경우 분할이 계속 하나씩 감소하는 경우로 $T(n) = T(n-1) + \Theta(n)$으로 전개시 $T(n) = \Theta(n^2)$가 된다. 1)

2.2 Deterministic Selection

2.1의 알고리즘은 확률은 적지만 최악의 경우 $O(n^2)$가 된다. partition을 할때 하나씩 되는 경우이다. 어떠한 상황에서도 $O(n)$을 유지하는 알고리즘이 Deterministic Algorithm이다. 적절한 pivot을 찾기 위해서 5개이 원소를 가진 subarray로 나눈다.. 각각의 집합에서 중앙값을 찾아 그 값을 pivot으로 삼고, 분할한다. 분할된 두그룹중 적절한 곳에서 재귀적으로 이를 반복하여 Selection하는 알고리즘이다. Pseudocode는 다음과 같다.

```
deter_select(input, p, r, k){
  if r - p <= 5 then
    A = sort(input[p:r]);
    return A[k];
  end if
  B = split(input, ⌈n/5⌉);
  int C[⌈n/5⌉];
  num = 0;
  for element in B
    Sort(elemennt);
    m = ⌊len(element)/2⌋ ;
    {C[num] = element[m]);}
  MoM = deter_select(C,0, ⌈n/5⌉ ,⌈n/10⌉);
  index = input.index(MoM);
  input[r] ⇔ input[index];
  q = partition(input,p,r);
  i = q- p +1;
  if k == i then return input[q];
  elif k > i then return deter_select(input, p, q-1, k);
  else return deter_select(input, q+1,r, k-i);
  end if
}
split(input, n){
  num = 0, A[n], B[5];
  for element in input
    B[num mod 5] = element;
    if num mod 5=4 then A[int(num/5)] = B;
    num ++;
  end for
  if num mod 5 != 0 then
    A[int(num/5)] = B[:num mod 5]
  end if
  return A
}
```

Deterministic Select의 Split, 각그룹의 중앙값 찾기의 구동 시간이 $\Theta(n)$, 중앙값의 중앙값 찾기가 $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$, 분할된 두 그룹 중 큰그룹이 선택되는 경우가 $T\left(\frac{7}{10}n+2\right)$가 된다.

즉, $T(n) \le T\left(\left\lceil \frac{n}{h} \right\rceil\right)+T\left(\frac{7n}{10}+2\right)+\Theta(n)$

$\le T\left(\frac{n}{5}+1\right)+T\left(\frac{7n}{10}+2\right)+\Theta(n)$

$T(k) \le ck$라고 가정하자. $n_0 \le k < n$

$T(n) \le c\left(\frac{n}{5}+1\right)+c\left(\frac{7n}{10}+2\right)+\Theta(n)$

$= c\left(\frac{9n}{10}+3\right)+\Theta(n) = cn - \frac{cn}{10}+3c+\Theta(n)$

$\le cn \quad \therefore T(n) = O(n)$, $T(n)= \Omega(n)$은 명백하므로 $T(n)= \Theta(n)$ 1)

2.3 Checker Program

본 과제에서는 2.1, 2.2를 구현한후, 출력값이 맞는지 체킹을 위한 체커 프로그램을 만들어야 했다. $O(n)$을 따르면서 Selection의 답을 구할 필요가 있다. Counting Sort에서 염감을 얻은 방식으로 Checker program을 구현하였고, Counting Sort와는 처럼 $O(M)$을 따라가는 것이아닌, 배열의 길이인 $O(n)$을 따라가게 구현 하였다.

대략적인 원리는 다음과 같다.

⓪ input의 Maximum 값을 확인한다. ($O(n)$)

M 값의 범위에 따라 두가지 케이스로 구현한다.

- $n \ge 2M$ … ⓐ
- $n < 2M$ … ⓑ

ⓐ인 경우

① size가 M이고, 원소가 0인 배열 A를 만든다 ($O(1)$)

② A[element] += 1 , (element는 input의 원소)를 한다. ($O(n)$)

③ A 값을 차례로 더해준다. $O(m)$

④ 더한 값이 k보다 크면 return

ⓑ인 경우(element를 e라하자)

① size가 2n 이고, 원소가 0인 배열 A,size n이고, 원소가 0인 배열 B 만든다. ($O(1)$)

② if element <= 2n: A[e] += 1 , (e는 input의 원소)

else: B[⌊n(e−2n)/m⌋].append(e) $O(n)$

③ A 값을 차례로 더해준다. $O(n)$

④ 더한 값이 k보다 크면 return

⑤ B의 각 원소는 array인데, 길이를 Sum에 더해주며 input의 k번째 원소가 속한 B의 원소를 각각 Quick Selection해준다. $O(n)$

ⓒ

*만약 음수가 있을경우 ②과정에서 -n보다 큰 것은 C[-element+1]에 저장하고, -n보다 작으면 D에 append 해준다.

D의 length를 확인해 D에 존재하면 D를 Qucikselection, C에 대해선 ③, ④처럼 진행하고, 답이 존재하면 return을 하고, 존재하지 않을 경우 ⓐ,ⓑ를 진행 시켜준다.

ⓐ에서 가장 최악의 경우를 보자. input이 공평하게 흩어져 있고, k가 n일경우 가장 worst case이다. 이경우 에도 ②가 n번, ③도 n번 실행되므로 ⓐ case의 경우 최악의 경우에도 $O(n)$이 성립된다.

ⓑ의 경우를 보자, ④에서 return될경우는 ⓐ와 같으므로, 더 뒤까지 가능 경우를 생각해보자.

가장 최악의 경우를 가정해보자.

A에 하나도 안 들어가고, 전부 B에 들어갔으며, m이 충분히 크고, 그 근처에 몰려있어 B의 한 원소에 전부 들어간 경우가 최악이다.

최악의 경우에 Quick Selection이 최악으로 돌면 $O(n^2)$가 된다.

하지만, 그 경우는 매우 드물고, QuickSelection에게 최악인 배열을 재외하면 안좋은 경우에도 $O(n)$에 수렴한다.

3. Result

3.1 Random과 Deterministic의 비

*test는 5번씩 진행했으며, 그래프에는 한장의 사진만 담았으나 결과는 거의 비슷했다.

10^7개의 array를 만들고, 각 원소는 임의의 0~10^8사이 값을 갖도록 하고, 시간을 측정하였다. 결과는 Fig1. , Fig2 과 같다.

Random은 $3.62*10^{-7}$을 기울기로 가지는 직선, deter는 $1.41*10^{-6}$을 기울기로 가지는 직선으로 수렴했다. 그래프에서 알 수 있듯이 deter은 크게 직선에서 벗어나지 않고 선형을 유지 함을

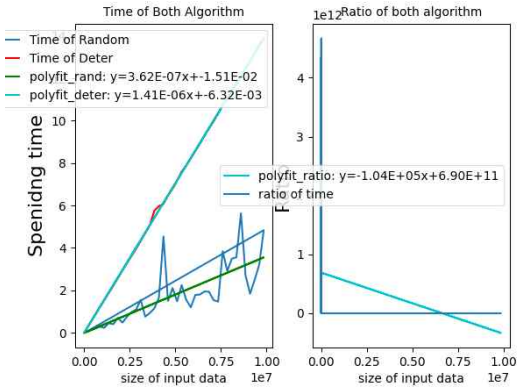

Fig1. Time of Random & Deterministic Selection Algorithm

볼 수있고, 우리가 유도한대로 잘 동작 함을 알 수 있다. Random은 역시 선택된 pivot에 따라 random한 time을 나타내지만 글래도 어느정도 선형 시간을 따라가며 조금식 변동을 보이는 것을 볼 수있다. 이번 케이스에서 deter: random비는 약$3.89:1$ 임을 볼 수있다.

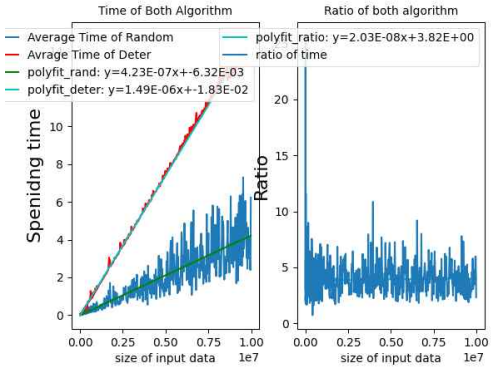

Fig2. Time of Random & Deterministic Selection Algorithm

같은 조건으로 다른 테스트셋을 만들어 진행한 결과가 Fig2. 이다. 같은 결과를 plot을 통해 확인 할 수 있고, Fig2.에서 기울기의비는 rand: deter = $3.52:1$이다.

이번에는 0~10^7까지 정렬된 input을 주었고, 결과는 Fig3 이다.

Deter와 random둘다 $O(n)$임을 볼 수 있다.

Deter:Random = $3:1$

여러 가지 테스트 케이스를 확인해본 결과

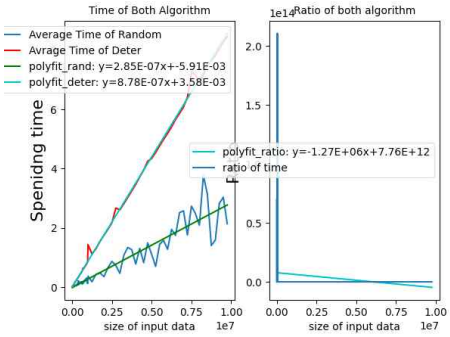

Fig3. Time of Random & Deterministic Selection Algorithm_sorted

Deter는 Random에 비해 약 3~4배 정도 느린 것을 볼 수 있다. 즉, 둘다 $O(n)$이지만, 계수가 3~4배정도 차이가 나는 것이라 생각된다.

3.2 Time of Checker.

3.1과 같은 Dataset으로 Checker의 시간을 측정한것은 다음과 같다.

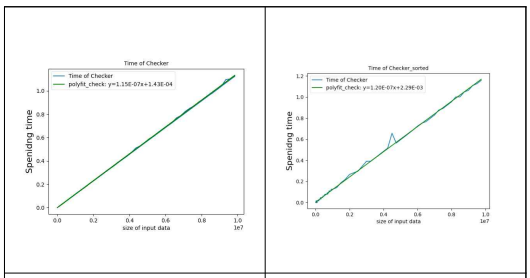

Fig4. Time of Checker_$(0 \sim 10^7)$	Fig5. Time of Checker(sorted)

매우 선형에 가까운것을 볼 수 있고. Random보다 더 기울기가 완만한것을 볼 수있다.(Random이 더 빠른 경우도 있었음)

4. Discussion

Checker의 ⓑ에서 ⑤번의 경우에서 Quick Selection으로 하면 최악의 경우에 $O(n^2)$을 가졌다. 하지만, 값을 빼줘 array B를 변형시킨 후, 다시 selectiont시켜 재귀를 해주면 결국엔 ⓐcase를 실행시키게 된다. 즉, 최악의 경우에도 $O(n)$의 실행속도를 보장할 수 있게 변형할 수 있다. 이것을 생각 했을때가 보고서 마감 직전이라 실제로 구현 못한 것이 아쉽다.

5. Reference.

1) 문병로. *관계 중심의 사고법 쉽게 배우는 알고리즘*. 한빛 아카데미.(2022). pp139-148