6주차 [과제]

이름 : 유재영

학번 : 20181650

제출일 : 2023.04.16

담당교수 : 최해철 교수님

과제. 평균 선형시간 선택 알고리즘 구현 및 분석

선택 알고리즘 중 평균 선형시간을 가지는 것을 구현하는 과제이다.

선택 알고리즘은 배열에서 `i`번째 작은 값을 찾는 과정으로, 퀵 정렬에서 사용된 `partition()` 으로 해결한다.

여기서 `partition()` 은 피봇이라는 것을 **기준(값)**으로 하여
피봇보다 작은 수는 왼쪽으로, 피봇보다 큰 수는 오른쪽으로 정렬하는 것이다.

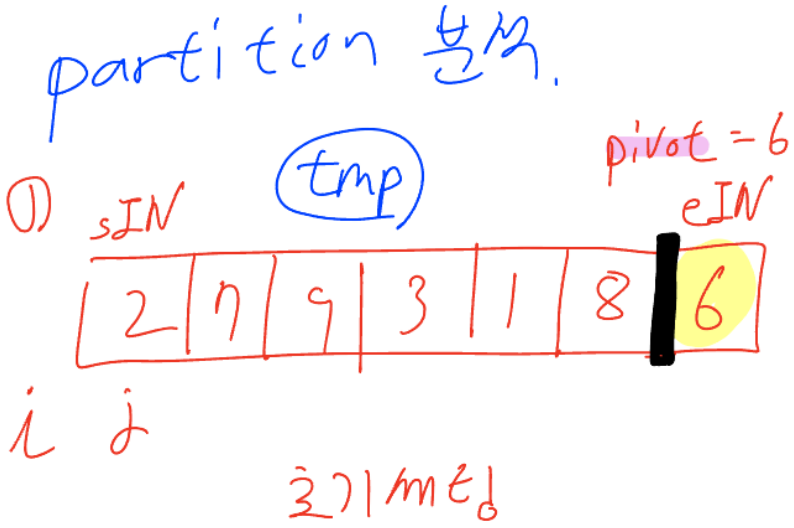
찾으려는 값보다 피봇이 큰 경우, 피봇을 기준으로 왼쪽 부분을 가지고 다시 `partition()`을 진행하고,

찾으려는 값보다 피봇이 작은 경우, 피봇을 기준으로 오른쪽 부분을 가지고 다시 `partition()`을 진행하고,

찾으려는 값과 피봇이 같은 경우, 그 값을 반환하면 된다.

`partition()` 의 반환값은 **피봇의 위치**인데, 이를 활용하여 선택 알고리즘을 구현 진행한다.

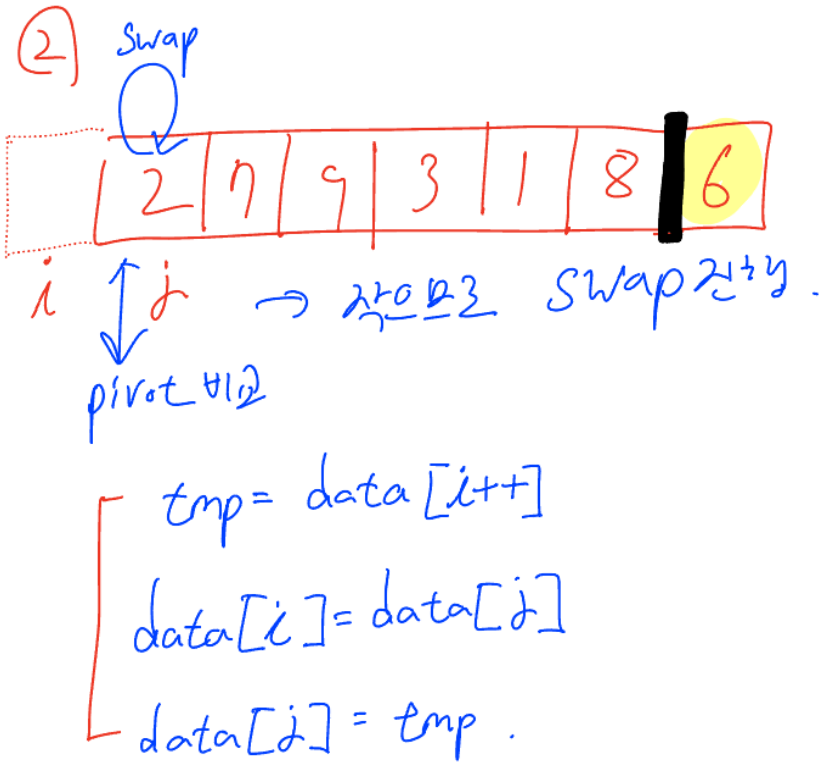
먼저 `partition()`을 분석해보려고 한다.



먼 초기 세팅을 아래와 같이 진행한다. (`data[]` 배열이 위와 같이 구성되어 있다고 가정한다.)

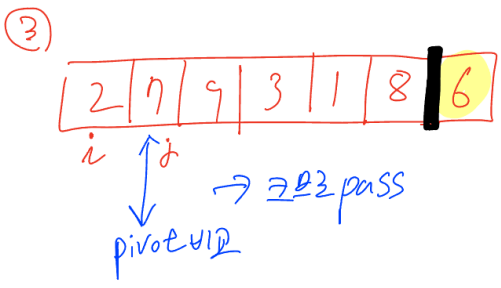
1. 시작 인덱스 번호 sIN, 마지막 인덱스 번호eIN으로 구성한다.
2. `int i = sIN -1 ;` 으로, `int j = sIN` 으로 초기화 한다.
3. `int pivot = data[eIN];` 으로 초기화 한다.

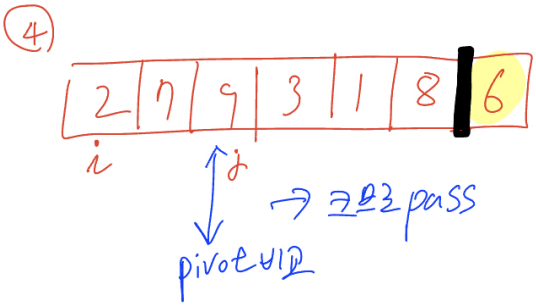
이후 **j 값을 기준으로** for문을 돌면서 `data[]` 배열을 분석 진행한다.



피봇보다 작은 값인 경우 스왑을 진행한다.

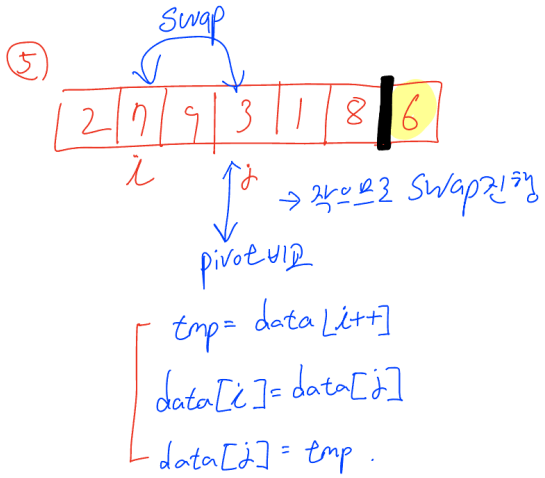
1. `tmp`에 `data[i++]` 저장
2. `data[i]`에 `data[j]` 저장
3. `data[j]`에 `tmp` 저장

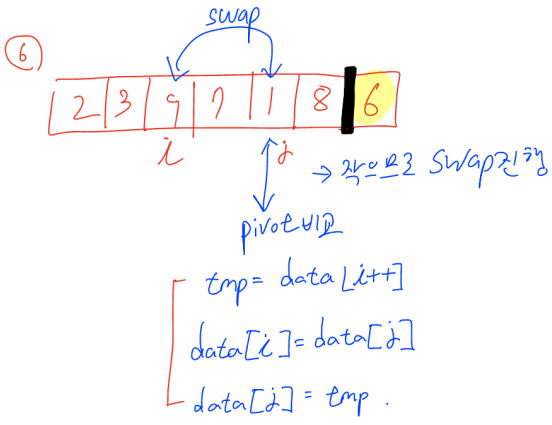


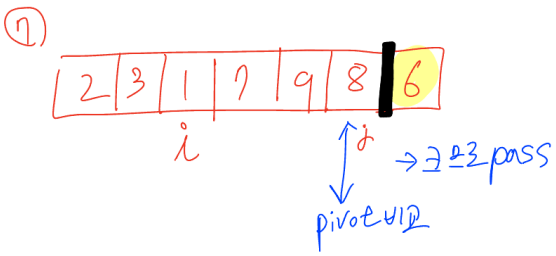


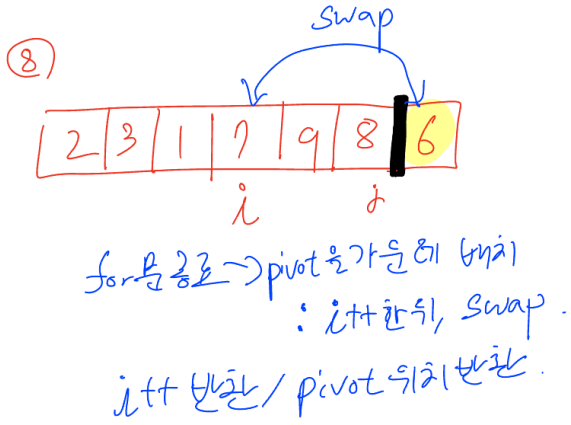
피봇보다 큰 값의 경우 넘어간다.

`partition()` 진행 과정을 그림으로 살펴보면,









최종 8번의 경우, `partition()`의 최종 부분이다.

마지막에 피봇 위치를 가운데로 변경하고, 해당 피봇의 위치값을 반환해 주는 것으로 끝이난다.

코드로 살펴보면 다음과 같다.

```java
// 가장 오른쪽 인덱스 값을 피봇으로 지정
int partition(int data[], int startIndexNum, int endIndexNum) {
    int pivot = data[endIndexNum];
    int tmp;
    int i = startIndexNum - 1;

    // 피벗보다 작으면 왼쪽으로, 피봇보다 크면 오른쪽으로 정렬
    for (int j = startIndexNum; j < endIndexNum; j++) {
        if (data[j] <= pivot) {
            tmp = data[++i];
            data[i] = data[j];
            data[j] = tmp;
        }
    }
    data[endIndexNum] = data[++i];
    data[i] = pivot;
    return i;
}
```

다음으로 `selectSort()` 를 작성하려고 한다.

퀵 정렬과 같은 방법이지만, 차이점은 **배열을 전부 정렬 진행하는 것이 아니다.**

찾으려는 값에 해당되는 범위에 대해서만 정렬하는 과정을 진행하는 것이기 때문이다.

코드는 아래와 같이 작성하였다.

```
25  int selectSort(int data[], int startIndexNum, int endIndexNum, int searchOrderNum) {
26
27      if (startIndexNum == endIndexNum) { // 시작 인덱스 번호와 마지막 인덱스 번호가 같은 상황 = 검색 하려는 값
28          return data[startIndexNum];
29      }
30
31      int pivotIndexNum = partition(data, startIndexNum, endIndexNum);
32
33      int pivotOrder = pivotIndexNum - startIndexNum + 1;
34
35      if (searchOrderNum > pivotOrder) {
36          // 피봇 기준 오른쪽 그룹에서 탐색
37          selectSort(data, startIndexNum: pivotIndexNum + 1, endIndexNum, searchOrderNum);
38      } else if (searchOrderNum == pivotOrder) {
39          // 찾으려는 값이 피봇이므로 반환
40          return data[pivotIndexNum];
41      } else {
42          // 피봇 기준 왼쪽 그룹에서 탐색
43          return selectSort(data, startIndexNum, endIndexNum: pivotIndexNum - 1, searchOrderNum);
44      }
45  }
```

25행에서 파라미터로 **검색할 배열, 시작 인덱스 번호, 마지막 인덱스 번호, 찾으려는 번호의 순서**를 받는다.

`selectSort()` 는 재귀 호출을 활용하여 진행하므로,

27행에서는 종료 조건으로 시작 인덱스 번호**(이하 sIN)**, 마지막 인덱스 번호**(이하 eIN)**가 같으면 종료한다.

이때, 반환하는 값은 `data[startIndexNum]` 즉, **찾으려는 값이다.**

31행에서는 피봇의 위치, 인덱스 번호를 받아 **pivotIndexNum**에 넣어둔다.

33행에서는 **피봇의 배열 내에서의 순위**를 찾는 과정이다. 해당 과정을 기준으로 아래에서 진행하게 되는데

35행은 피봇의 순위가 찾으려는 순위보다 **작은 경우**이므로, **피봇 기준 오른쪽 그룹**에서 다시 탐색을 진행한다.

38행은 피봇의 순위가 찾으려는 순위와 **같은 경우**이므로, 해당 값을 반환한다.

41행은 피봇의 순위가 찾으려는 순위보다 큰 경우이므로, **피봇 기준 왼쪽 그룹**에서 다시 탐색을 진행한다.

→ 이때 함수의 이름이 틀렸다는것을 깨닫고 함수이름도 `selectFind()`로 수정하였다.

위와 같이 작성한 후 메인문을 돌렸을때 다음과 같은 결과가 나왔다.

```
/Users/yujaeyeong/Developments/algorithm/univ-algorithm/230411/cmake-build-debug/230411
랜덤 수의 개수를 입력 : 5
찾으려는 순번 입력 : 3
검색하는 배열 : 3 7 8 4 0
배열의 3번째 값은 '1'입니다.
```

~~비상이다~~ 무언가가 틀린듯 하다..

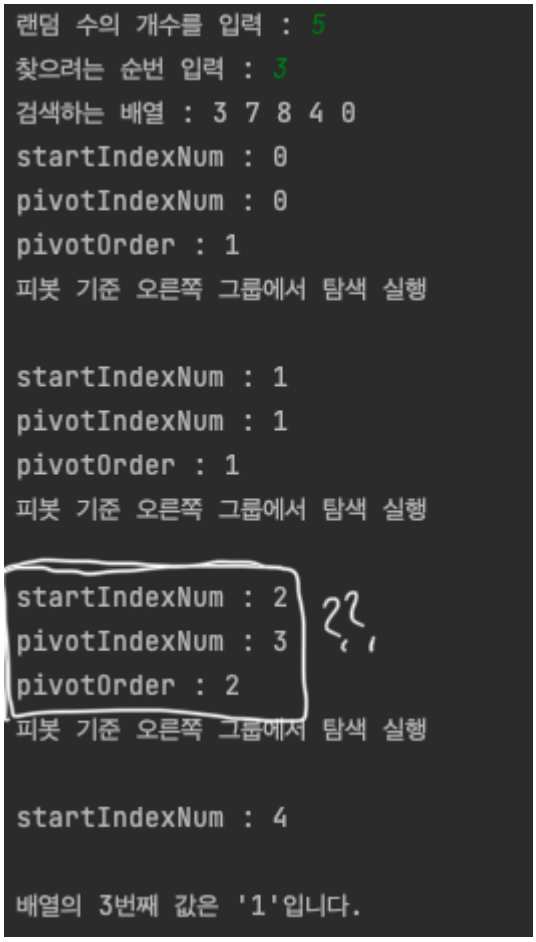
`partition()` 은 퀵정렬에 대입해서 실행하여도 아무 이상없이 돌아갔다.

`selectFind()` 부분 문제인듯 싶어 아래와 같이 수정 후 검색을 진행하였다.

`data[]` 에는 오류가 났었던 검색하는 배열을 직접 넣어주었다.

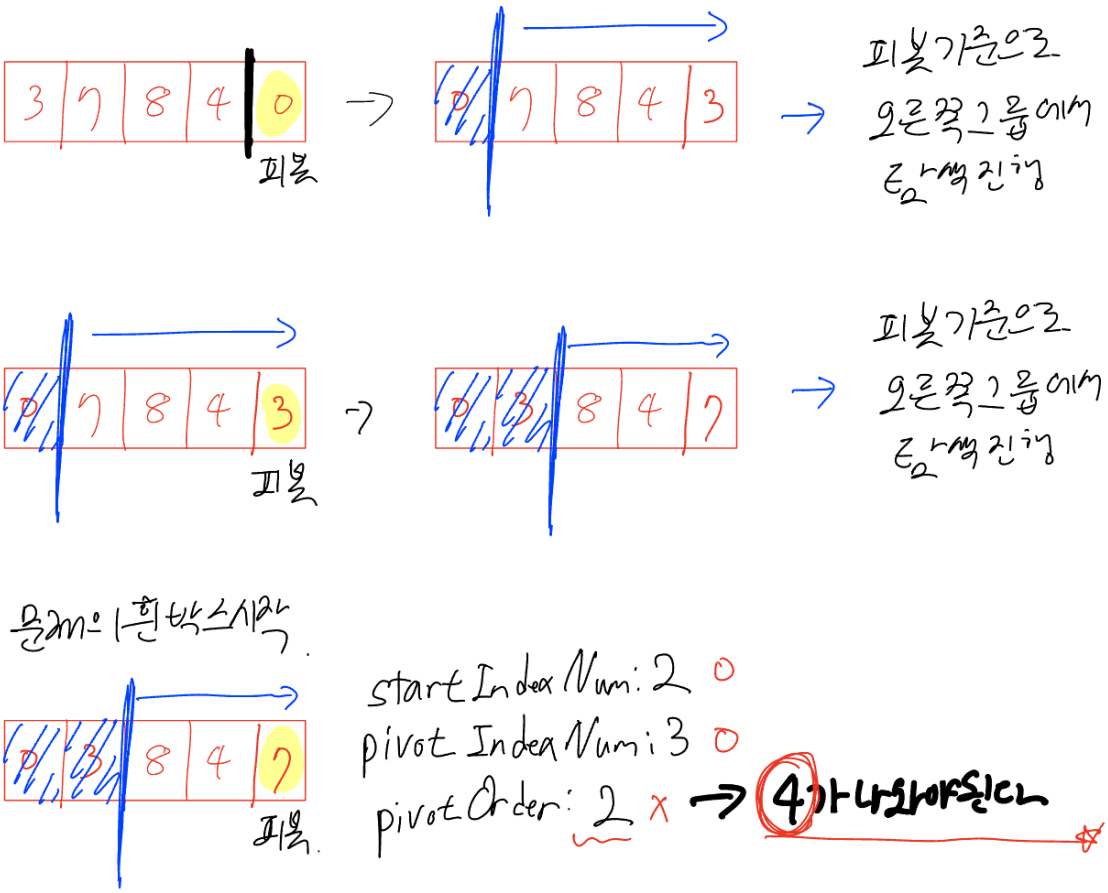
```
25  int selectFind(int data[], int startIndexNum, int endIndexNum, int searchOrderNum) {
26
27      printf("startIndexNum : %d\n", startIndexNum);
28      if (startIndexNum == endIndexNum) { // 시작 인덱스 번호와 마지막 인덱스 번호가 같은 상황 = 검색 하려는 값
29          return data[startIndexNum];
30      }
31
32      int pivotIndexNum = partition(data, startIndexNum, endIndexNum);
33      printf("pivotIndexNum : %d\n", pivotIndexNum);
34      int pivotOrder = pivotIndexNum - startIndexNum + 1;
35      printf("pivotOrder : %d\n", pivotOrder);
36
37      if (searchOrderNum > pivotOrder) {
38          // 피봇 기준 오른쪽 그룹에서 탐색
39          printf("피봇 기준 오른쪽 그룹에서 탐색 실행\n\n");
40          selectFind(data,  startIndexNum: pivotIndexNum + 1, endIndexNum, searchOrderNum);
41      } else if (searchOrderNum == pivotOrder) {
42          // 찾으려는 값이 피봇이므로 반환
43          printf("찾으려는 값이 피봇이므로 실행\n\n");
44          return data[pivotIndexNum];
45      } else {
46          // 피봇 기준 왼쪽 그룹에서 탐색
47          printf("피봇 기준 왼쪽 그룹에서 탐색 실행\n\n");
48          return selectFind(data, startIndexNum,  endIndexNum: pivotIndexNum - 1, searchOrderNum);
49      }
50  }
```

출력은 아래와 같다.



결과를 보니, 사각형을 친 부분에서 이상하게 출력되는 것 같았다.

따라서 해당의 왜 그렇게 출력이 되는지 분석을 진행해보았다.



1이 나오는게 말이 안된다.....

```
38      int pivotIndexNum = partition(data, startIndexNum, endIndexNum);
39      printf("pivotIndexNum : %d\n", pivotIndexNum);
40      int pivotOrder = pivotIndexNum + 1;
41      printf("pivotOrder : %d\n", pivotOrder);
```

혹시나 해서, pivotOrder도 연산이 이상한것 같아 수정하였다.

pivotOrder은 사실 pivotIndexNum+1만 해도, 상관없기에 다음과 같이 수정하였으나 여전히 같다.

그러던 와중, **예상치 못한 곳에서 문제점을 발견했다.**

```
if (searchOrderNum > pivotOrder) {
    // 피봇 기준 오른쪽 그룹에서 탐색
    printf("피봇 기준 오른쪽 그룹에서 탐색 실행\n\n");
    selectFind(data, startIndexNum: pivotIndexNum + 1, endIndexNum, searchOrderNum);
} else if (searchOrderNum == pivotOrder) {
```

return을 빼먹은 것이다.

왜 1이 나온지는 이유를 모르겠으나, return문이 없어서 값이 이상하게 출력된 것으로 보인다.

```
startIndexNum : 2
시작 인덱스 번호와 마지막 인덱스 번호가 같은 상황 = 검색 하려는 값 실행
data[startIndexNum] : 4
0 3 4 7 8

배열의 3번째 값은 4입니다.
```

정상적으로 출력된다!

다시 main()문을 돌려보게되면,

```
/Users/yujaeyeong/Developments/algorithm/
랜덤 수의 개수를 입력 : 10
찾으려는 순번 입력 : 3
검색하는 배열 : 49 91 6 29 87 45 59 65 22 94

배열의 3번째 값은 29입니다.

Process finished with exit code 0
```

```
/Users/yujaeyeong/Developments/algorithm
랜덤 수의 개수를 입력 : 10
찾으려는 순번 입력 : 3
검색하는 배열 : 6 61 43 59 48 0 9 45 97 97

배열의 3번째 값은 9입니다.

Process finished with exit code 0
```

```
/Users/yujaeyeong/Developments/algorithm/
랜덤 수의 개수를 입력 : 10
찾으려는 순번 입력 : 3
검색하는 배열 : 23 92 78 99 79 3 79 29 23 13

배열의 3번째 값은 23입니다.

Process finished with exit code 0
```

```
/Users/yujaeyeong/Developments/algorithm/
랜덤 수의 개수를 입력 : 10
찾으려는 순번 입력 : 3
검색하는 배열 : 35 82 82 98 83 14 36 7 74 99

배열의 3번째 값은 35입니다.

Process finished with exit code 0
```

정상적으로 출력된다.

앞으로 코드 작성할 때, **조금 더 꼼꼼히 봐야겠다**라는 반성을 하였다.