코멘토 진로부트캠프 1주차 과제
<< 실전에 활용할 수 있는 알고리즘 개발 역량 강화하기 >>
서울대 대학원 참여자
화학부 이수민

#1.

구현코드는 압출 폴더내 sort.cpp에 있습니다.
quick sort에서 pivot을 어디로 잡는지에 따라 차이가 났는데, ascending/descending order의 경우 pivot을 양 끝으로 할 경우 매 함수 호출에서 나머지 요소들이 한쪽으로만 할당되기 때문에 worst case 시간이 걸렸습니다. 중간 값을 고를 경우 divide and conquer 효과가 나타나 빠르게 정렬할 수 있었습니다.

```
(결과 예시)
$./sort
ascending order
insertion sort time: 0.010278s
merge sort time: 0.193083s
quick sort time: 0.10998s

descending order
insertion sort time: 1090.07s
merge sort time: 0.168305s
quick sort time: 0.098142s

random order
insertion sort time: 532.357s
merge sort time: 0.258753s
quick sort time: 0.216659s
```

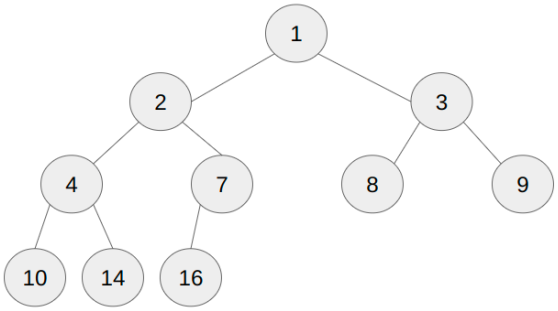
#2.

* priority queue
자료구조의 한 종류로, 모든 항목에 우선순위(key)가 부여되어 있는 1D 구조입니다. 새롭게 요소를 추가하는 enqueue, 가장 우선순위가 높은 것을 반환하는 peek, 우선순위 높은 것을 반환하고 quueue에서 지우는 deque 동작을 지원합니다. priority queue를 구현하기 위해 복잡도가 O(logN)인 heap 이 주로 이용됩니다.
peak: root heap 반환 O(1)
enqueue: heap 끝에 새 요소를 추가하고 heap 조건이 만족되도록 부모노드와 값을 바꿉니다.O(logN)
dequeue: max heat을 반환하고, heapifiy를 통해 남은 노드들로 heap으로 만듭니다. O(logN)
ref: https://yoongrammer.tistory.com/81

* heap sort: heap 구조를 활용하여 sorting 하는 방법입니다. max_heap을 만든 후 최대요소를 마지막 요소와 교환하고, heap에서 마지막 노드를 없앤 후 max_heapifiy를 통해 다시 max heap으로 만들어 주는 것을 계속 반복합니다. build_max heap는 devide and conquer 접근방법으로 해결하며, 재귀함수 형태로 구현됩니다. 각 함수 호출에서는 subtree가 max heap 임을 가정하고 root 요소에 대해 max heap을 만족시키도록 재구성하는 작업이 수행됩니다.
ref: https://gmlwjd9405.github.io/2018/05/10/algorithm-heap-sort.html,
opencourseware 6.006-Introduction to algorithm, lecture notes (lecture4).

* heap sort의 장점: 복잡도가 O(N*log(N))으로 효율적이며, 다른 정렬의 경우 최댓값, 최솟값을 구하기 위해 전체 서열에 대해서 정렬을 수행해야 하지만 heap sort는 상위 몇개의 값만 부분적으로 추출할 수 있다는 장점이 있습니다. 소수의 상위값만 필요한 경우에는 훨씬 빠른 속도로 답을 구할 수 있습니다.
ref: https://gmlwjd9405.github.io/2018/05/10/algorithm-heap-sort.html

#3.



#4.

* counting sort
카운팅 정렬은 요소값을 배열의 index와 관련지어 counting한 후 차례대로 삽입하여 정렬하는 방법입니다.

1) min, max 값을 찾고, max-min+1 크기 만큼의 배열을 만들어 0으로 초기화 합니다.
2) 배열을 순회하면서 요소의 빈도를 축적해 계수합니다.
3) 계수해둔 배열을 순회하면서 해당 값을 빈도만큼 삽입합니다.

O(k+n) 복잡도를 가지며 k는 배열 요소의 최댓값, n은 배열의 크기를 나타냅니다.
가장 효율적일 때는 n=k인 경우로, 반복이 없는 경우 linear time에 정렬할 수 있습니다.
반대로 k가 매우 크거나 요소간 차이가 크면 k+1 크기 만큼의 배열을 만들어야 한다는 점에서 메모리 낭비가 심하다는 단점이 있습니다.

* radix sort
기수정렬은 1의 자리수부터 가장 큰 자리수까지 비교하면서 정렬하는 방법입니다.

1) max 값을 찾아 max 값의 자릿수만큼만 아래 정렬과정을 반복합니다.
2) counting sort로 특정 자리수만 고려하여 sorting 합니다. O(N)

O(n) 복잡도로 매우 빠르다는 장점이 있지만, 데이터 타입이 일정해야 하고, 양수 음수 나누어 비교해야 하며 추가적인 메모리공간이 필요하다는 제한이 있습니다.

ref: https://velog.io/@wjdqls9362/Algorithm-%EC%A0%95%EB%A0%AC-Radix-sort-Counting-sort

#5.

peakfinder.cpp에 2D peak finder를 구현하였습니다. 예시 인풋으로 input.txt, input2.txt를 만들어 확인하였습니다.

```
(결과예시)
$./pfinder
Please enter the file name (e.g. input.txt):
input.txt
10 8 10 10
14 13 12 11
15 9 11 21
16 17 19 20
```

```
Peak value: 21
Peak position: (3, 4)
```

#실전문제풀이

solution.cpp에 구현하였습니다. 첫번째 숫자에 따라 인풋으로 받은 정수를 분류하였고, 각 집합 안에서 merge sort를 이용해 자리수가 적고, 자릿수가 같을 때는 큰 숫자가 먼저 오도록 정렬하였습니다.
10등분으로 나누면 O(N+0.1N*log(0.1N)) 로 O(N*log(N)) 보다 작기 때문에 나누었습니다. 0과 1000은 따로 분류하여 1000이 먼저 오고 그 다음 0이 오도록하여 마지막에 string으로 합쳐주었습니다.

```
(결과예시)
$./compile.sh
$./realtest
Please enter a sequence of zero or positive integers separated by comma:
1) each interger should be less than 1,000
2) the lenght of the sequence should be less than 1000,000
e.g.: [0,12,245,38]
[0,12,99,40,1000]
input seq:
0 12 99 40 1000
output: 99401210000
```