COMP319 Algorithm 1, Spring 2020
Programming assignment 3

Instructor: 장길진(gjang@knu.ac.kr, Tel 053-950-5517

TA: 박진희 (pjhdrm@gmail.com, Tel 053-940-8617)

General information

1. 채점기준:
    a. 제출했을 경우: 10% 기본점수
    b. 첫줄의 주석으로 학번과 이름 제대로 명시: 10%
    c. 정확도 및 실행시간 40% - 10 개 정도의 예제를 돌려보고 평균 실행시간을 측정하여 상대 평가. ~~*출력은 median 만 허용하며,*~~ 틀렸거나 compile 이 안 될 경우 그 예제에 대하여 0 점
    d. 화면 출력: **hw3_1.c/hw3_2.c: sorting 결과, hw3_3.c: median + sorting 결과**, template code 에 나와 있는 "fprintf(stdout, …)"만 남겨 두면 됨
    e. 코드 reading 점수 40% - 문제에서 요구하는 조건들을 만족시켰는가?
2. 감점기준
    a. 지각제출은 1 시간마다 10% 감점이 있습니다.
        i. LMS 기준 00:00 ~ 00:59 -10%
        ii. LMS 기준 01:00 ~ 01:59 -20%
        iii. …
        iv. LMS 기준 09:00 ~ 09:59 -100%
    b. Cheating (카피):
        i. 교수/조교/튜터가 같이 보고 평가하여 다수결로 결정
        ii. 동일한 코드가 나오면 "copied"/"being copied" 여부에 관계없이 해당 점수 0 점
        iii. Internet 에서 찾거나 책에 있는 코드를 사용하였을 경우 제출한 코드 안에 주소나 책 등을 명시할 것. 예를 들어 heapify() 나 heapsort() 같은 함수들은 워낙 많이 구현되어 있기 때문에, 일부러 다르게 쓰기도 힘들다. 이럴 경우, 자신이 참조한 코드를 명시해 주면 완전히 동일하더라도 cheating 으로 판단하지 않습니다. 즉, 많이 알려진 코드라면 출처를 명시하면 다른 학생들과 같아도 상관없습니다.
        iv. 출처가 '**친구**', '**선배**' 등은 허용하지 않습니다.
3. 제출물
    a. Submit only .c files, nothing else
    b. No VC++ project, no executable, no data (C 파일 이외에 다른 파일을 제출하면 감점)
    c. 소스 코드들은 하나로 묶어서 hw3.zip 파일로 제출함. 파일들을 따로 제출하면 채점이 매우 어렵습니다.
    d. 주어지는 template (*_template.c)은 '참고용' 입니다. 입출력만 적합하게 구현하면 본인의 취향에 따라 얼마든지 바꾸어도 괜찮습니다.
4. Compilation:
    a. Your code will be compiled by gcc 5.4.0 under Ubuntu linux
    b. In Windows environment, code::blocks is recommended (http://www.codeblocks.org/downloads)
    c. Linux 와 code::blocks 는 같은 gcc compiler 를 쓰기 때문에 (이론적으로는) 결과가 같습니다. Compiler version 은 조금 다를 수 있습니다.

문제 설명

1. **(15%) Heap sort using MaxHeapify** (max heap 을 이용하여 오름차순 정렬을 구현한다.)
    a. **구현할 것들:**
        i. void MaxHeapify(int arr[], int n, int i)
            1. array 를 max heap 구조로 바꾸는 함수를 구현한다.
            2. Textbook 은 1-based indexing (즉 array 가 1 부터 시작)을 쓰고 있는데, 0-based indexing 을 써도 관계없음(일반적인 C 의 array 와 같이 0 부터 시작). 다만 그에 맞춰서 heapify 를 다르게 작성해야 한다.
        ii. void MaxHeapSort(int arr[], int n)
            1. 위의 MaxHeapify 함수를 이용하여 heap 을 구성하고, 최대값을 배열의 마지막으로 보내고, 바뀐 subtree 에 대하여 MaxHeapify 를 적용하는 방식으로 구현한다.
    b. 예시: (입력) **input_____.txt**, (코드 template) **hw3_1_template.c**
    데이터의 입력은 fscanf 함수만을 사용(template 코드에 있음. 채점을 위하여 바꾸면 안 됨)
    c. **제출물**: **hw3_1.c**
    d. **코드활용**: 위의 함수들은 워낙 많이 구현되어 있기 때문에, 이미 구현되어 있는 코드를 활용하는 것을 권장함. 단, 반드시 자신이 참조한 코드의 출처(웹주소 등)를 명시해야 한다. 직접 처음부터 작성하였을 경우 명시할 필요 없고, 좀더 높은 점수를 받을 가능성도 있지만 코드의 유사성에 따라 copy 판정을 받을 수 있는 것을 유의한다.

2. **(25%) Heap sort using MinHeapify** (min heap 을 이용하여 오름차순 정렬을 구현한다.)
    a. **구현할 것들:**
        i. void MinHeapify(int arr[], int n, int i)
            1. array 를 min heap 구조로 바꾸는 함수를 구현한다.
        ii. void MinHeapSort(int arr[], int n)
            1. 위의 MinHeapify 함수를 이용하여 오름차순 정렬을 구현한다. min heap 의 root 는 minimum 이기 때문에 일반적인 heap sort 와는 다르다(MaxHeapify 를 사용하면 안 됨)
    b. 예시: (입력) **input_____.txt**, (코드 template) **hw3_2_template.c**
    데이터의 입력은 fscanf 함수만을 사용(template 코드에 있음. 채점을 위하여 바꾸면 안 됨)
    c. **제출물**: **hw3_2.c**
    d. **코드활용**: hw3_1.c 의 MaxHeapify()를 수정하여 MinHeapify()를 구현한다. 본인이 작성한 MaxHeapify()와 유사하면 copy 판정에서 자유롭다. MinHeapSort()는 꼭 직접 작성하여야 한다(download 허용하지 않음).

3. **(60%) Heap for finding median and sort** (Median 을 찾기 위한 Heap 구현 및 이를 이용한 sort)
   Median value 를 쉽게 찾기 위해서 다음과 같은 heap 구조를 구현하고 이를 이용하여 오름차순 정렬을 하는 코드를 작성한다.
   a. **조건:**
      i. 전체 data 들은 두 개의 heap L 과 R 에 저장된다.
      ii. all values in L < any value in R
      iii. 전체 item 의 개수가 N 개일때
         1. N 이 홀수: numbers of items in L = N/2+1, and in R = N/2 (정수 나눗셈)
         2. N 이 짝수: numbers of items in L = number of items in R = N/2
            즉, L 에 있는 원소의 개수가 R 의 원소 개수와 같거나 하나 더 많다
      iv. L 은 max heap, R 은 min heap 을 유지한다. 그러면 median 은 언제나 root(L) 이다 (다른 문서들을 보면 N 이 짝수일때는 N/2 번째와 N/2+1 번째의 평균으로 찾는다고 되어 있는데, 여기서는 문제를 간단히 만들기 위해 그냥 root(L)로 하기로 합니다. 그리고 all values in L <= root(L) 을 만족하기 때문에 median 의 정의에 부합됨.
      https://www.quora.com/How-do-you-find-the-median-from-a-set-of-even-numbers
      https://www.dkfindout.com/us/math/averages/more-about-medians/)
      v. 작성한 'heap for median'을 오름차순 정렬로 바꾼다.
   b. **구현할 것들**:
      i. 위와 같은 구조에 하나의 item (value)를 집어 넣는 다음의 함수를 작성한다
      void Heap4Median_AddItem(int L[], int *p_numL, int R[], int * p_numR, int value)
         1. **입력**: 크기가 N 인 배열, A[], N 로 주어짐
         2. **출력**: 크기가 *p_numL, *p_numR 인 두개의 heap 구조의 배열, L[], *p_numL, R[], *p_numR 에 저장하여 return 함.
         3. Pointer 변수 p_numL 과 p_numR 은 L 과 R 의 개수가 달라질 수 있으므로 pointer 로 주고 받아야 한다(Input & Output 동시에)
         4. root(L), root(R), value 사이의 관계와 *p_numL, *p_numR 에 따라 다르게 처리해야 한다.
         5. 일반적인 경우 Heapify 는 root 값이 바뀌었을 경우에만 해당됩니다. 따라서 heap 에 원소를 맨 뒤에 추가하는 함수가 필요합니다(heap_insert_item or heapify_up)
            참고: http://pages.cs.wisc.edu/~mcw/cs367/lectures/heaps.html
      ii. Unsorted array 입력에 대하여 L 과 R 을 생성하는 다음의 함수를 작성한다.
      Heap4Median_Build(int A[], int N, int L[], int *p_numL, int R[], int *p_numR)
         1. **입력**: 크기가 N 인 배열, A[], N 로 주어짐
         2. **출력**: 크기가 *p_numL, *p_numR 인 두개의 heap 구조의 배열, L[], *p_numL, R[], *p_numR 에 저장하여 return 함.
         3. A 는 size N array, L 과 R 은 size N/2+1 array 로 할당하며, 초기개수로 *p_numL = *p_numR = 0 (즉 L 과 R 은 empty array)
         4. *p_numL 과 *p_numR 은 "a. 조건-iii" 에 의해 정해진다.
         5. **방법 1**: 모든 N 개의 원소에 대하여 Heap4Median_AddItem() 을 실행한다.
         6. (*optional*) 대안: Heap4Median_AddItem() 이 복잡하기 때문에 다른 방법으로 만들어 볼 수도 있다.

a. 예를 들어 전체를 가지고 max heap 을 만들고 하나씩 빼면서 L 을 만들고 나머지로 R 을 만들거나, 대충 heap 2 개를 만들고 L 과 R 로 재조정 등 몇 가지 다른 방법들이 있을 수 있다.
b. 시간 복잡도를 고려하여 다양한 방법을 시도해 볼 수 있으며, 위의 방법 1 보다 더 좋은 알고리즘을 찾으면 추가 점수 부여함

c. 생성된 heap4median 을 이용하여 오름차순 정렬 array 를 만드는 다음의 함수를 작성한다.
Heap4Median_Sort(int L[], int numL, int R[], int numR, int A[], int *p_N)
1. **입력**: 크기가 numL, numR 인 두개의 heap 구조의 배열, L[], numL, R[], numR 로 주어짐
2. **출력**: 크기가 *p_N 인 배열, A[], *p_N 로 return 함
3. 최종적으로 *p_N=numL+numR. 배열 L 과 R 는 변경되어도 된다.
4. **방법**: L 을 heap sort 하고, R 을 heap sort, 그리고 A 에 복사. 또는 L 과 R 에서 값을 하나씩 빼서 A 에 하나씩 집어넣기

d. 예시: (입력) **input_____.txt**, (코드 template) **hw3_3_template.c**
데이터의 입력은 fscanf 함수만을 사용(template 코드에 있음. 채점을 위하여 바꾸면 안 됨)

e. **제출물**: **hw3_3.c**

f. **코드활용**: hw3_1.c 의 MaxHeapify(), MaxHeapSort()와 hw3_2.c 의 MinHeapify(), MinHeapSort()를 활용하는 것을 적극 권장한다(필수는 아님). Heap4Median_AddItem(), Heap4Median_Build(), Heap4Median_Sort() 는 꼭 직접 작성하여야 한다(download 허용하지 않음).

(끝)