CSE100 Algorithm Design and Analysis Lab #4
Spring semester 2013 Miguel Á. Carreira-Perpiñán

Sorting: HeapSort and QuickSort

Description In this assignment you are requested to implement two different algorithms for sorting elements in an array. Please make sure that you create one program for each algorithm.

1. Implement the heapsort algorithm for sorting an array of integers.
   Testcase number to use with `grade_me`: 4.
2. Implement the quicksort algorithm, in which you select your pivot point to be the last element in the array (or sub-array). See section 7.1 of the book.
   Testcase number to use with `grade_me`: 4.

Input structure The input may contain multiple cases. Each case starts with an integer number which indicates the number of elements to be sorted. Then, the elements follow, one per line. A sequence of 0 elements indicates the end of the input, and produces no output. You can assume the input is correctly structured (i.e., no data are missing).

Output structure Output the sorted sequence one element per line case by case until you hit 0. Do not insert spaces at the beginning or at the end of any element.

Examples of input and output

Testcase 4: input

```
6
3
6
23
76
4
56
0
```

Testcase 4: output

```
3
4
6
23
56
76
```

Optional question 1 The time complexity for quicksort depends on which pivot is selected. Rewrite your quicksort to use a random element of the array as your pivot (book section 7.3).

Optional question 2: experimental runtimes You have to compare the runtime for different sorting algorithms (heapsort and quicksort) on various test cases. In this and future labs, we will use the following setup, which the TA will demonstrate (it looks complicated but in practice it is very easy):

- *Location of supporting files*: all the files we provide below are contained in the directory `engapps00.ucmerced.edu:/home/mvladymyrov/bin/lab03/`.
- *Generating test cases*: use the C function (provided by us) `rndarray(s,k,n,seed)`, which creates a random array of integers, as follows:
  - `s` (char): array sorted in ascending (`‘a’`) or descending order (`‘d’`), or unsorted (`‘u’`).
  - `k` (int): the array values will be in `0..k`. Note: for this lab, fix $k = 10^8$.
  - `n` (int): the size of the array to be generated.
  - `seed` (int): the seed to initialize the pseudorandom number generator. Note: fix this to your student id# for all your experiments.

  The function `rndarray` needs the auxiliary object file `xsort.o` (provided by us).
- *Measuring run times*: you can obtain the run time of your algorithm by bracketing your sorting function with calls to the C function `gettimeofday()`. Use the template C program `SortTime.c` (provided by us) for this purpose.
- *Plotting run times*: use the Matlab function `PlotRuntime.m` (provided by us). All you need to do is type your run times into a matrix T of $N \times D$ (where N is the number of array sizes tried and D the number of conditions tried) and run

`PlotRuntime.m`. Use `NaN` for missing entries (for example, when the practical runtime is expected to be very long). The figure will appear on screen and on a file `runtime.eps`.

What you have to do:

1. Merge your sorting C function for heapsort into `SortTime.c` and build the executable following the instructions inside `SortTime.c` (if your sorting function is written in C++, convert it into C style for the merge).

2. Generate random arrays of different sizes n and types (ascending, descending, etc.) with runs of the form `SortTime s k n seed`, and write down the sort running time $T(n)$ printed by `SortTime` (in seconds).
   For this lab, we suggest you try $n \in \{10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8\}$ (total 7 different array sizes), for each of `s={'a','d','u'}` (total 3 different array types). This will result in a table T of 7×3.

3. Plot the run times with `PlotRuntime.m` given your table T. This will show 3 curves $T(n)$ (i.e., as a function of the array size n) for each of `s={'a','d','u'}`, and save them to a file `runtime.eps`. Rename this file `runtime_heapsort.eps`.

4. Write (in a file `runtime_heapsort.eps`) a concise report describing your results. Consider the questions: how do the curves compare with the asymptotic running times in the best, worst and average case for the algorithms? What happens for very small or very large n?

5. Repeat for the other algorithm (quicksort).

6. Finally, upload your code and `.eps` and `.txt` files.

Your solutions Before leaving the lab, submit a zipped tar archive of your program through the assignments page of UCMCROPS. Please use your UCMNetID as the filename for the zipped tar archive. Be careful since UCMCROPS strictly enforces the assignment deadlines (deadlines will be every lab date at either 4:20pm or 7:20pm depending on your lab session).