Assignment 2: Heaps  
CS 758/858, Spring 2016  
Due Monday, February 8 by 12:40pm

Implementation

The skeleton code on the course web page is the start of a program to add floating point numbers in a way that attempts to preserve accuracy: always add together the two smallest values. Complete the program by implementing an improved version that uses a heap instead of an $O(n)$ scan. Then measure the resulting performance improvement.

The program reads floats from standard input, one number per line. Its output is the sum of the floats and the time it took the program to run.

The skeleton code implements three sample algorithms:

- **seq** Performs a simple sequential sum of the numbers in the order in which they were given.
- **sort** Sorts the numbers in increasing order and then sums them sequentially.
- **min2_scan** Perform a linear scan of the numbers to find the minimum two values. These values are removed and their sum is added back into the set of values. The process then repeats.

Your job is to implement the **heap** method, which is just like **min2_scan** but using a heap for faster performance.

Testing

On the course web page, we supply some utility programs and input files. Most of the programs we distribute in this class will tell you their command-line arguments if you run them with the **--help** option.

- **add** is your program. If you wanted to test your algorithm on a given set of numbers, it would look like:

  ```
  ./add heap < 100_floats
  ```

- **add-harness** runs your program, checks its output, and optionally displays a plot of the performance. For example:

  ```
  add-harness -d -a seq -a sort -a min2_scan 2000 4000 8000 16000
  ```

  will sum 2,000, 4,000, 8,000 and 16,000 random sets of floats using the seq, sort and min2_scan algorithms (five times each by default). Two plots will then be displayed: one showing time and one showing accuracy (the difference between the true sum computed using slow infinite-precision arithmetic and the sum from the given algorithm). If you are running the harness on a system without an X display (e.g. agate) then you may use the **-o <filesuffix>** option to output the plots to files instead (suffix should end in .pdf, .ps or .png).

Written Problems

1. Briefly summarize which parts of your program are working or not. Include transcripts or plots showing the successes or failures. Is there anything else that we should know when evaluating your implementation work?

2. What can you conclude about the performance of the various algorithm implementations from your experiments? Please include your plots in your submission (electronic and hardcopy).
3. Here is some psuedo-code:

**match** (x, y)
1. if x and y are the same, return(TRUE)
2. else return(FALSE)

**contains** (list, x)
3. if list = NULL, return(FALSE)
4. else if match(list.data, x), return(TRUE)
5. else return(contains(list.next, x))

**index** (list, x)
6. return index-helper(list, x, 0)

**index-helper** (list, x, i)
7. if list = NULL, return(-1)
8. else if match(list.data, x), return(i)
9. else return(index-helper(list.next, x, i + 1))

**remove** (list, x)
10. if list = NULL, exit(ERR)
11. else if match(list.data, x), return(list.next)
12. else
13. list.next ← remove(list.next, x)
14. return(list)

slow-remove-all (list, x)
15. while contains(list, x)
16. if index(list, x) ≥ 0
17. list ← remove(list, x)

(a) Imagine that l is a list that contains a 1,000 times followed by b 1,000 times. If we run slow-remove-all(l, b), exactly how many times will match be executed?

(b) What is the worst-case time complexity of slow-remove-all?

(c) Give a better implementation of slow-remove-all. What is its complexity?

4. Problem 3–2 from CLRS.
5. Exercise 6.1–3 from CLRS.
6. Exercise 6.1–4 from CLRS.
7. Exercise 6.5-5 from CLRS.
8. Exercise 6.5-9 from CLRS.
9. (858 only) Exercise 6.2–6 from CLRS
10. (858 only) Exercise 6.4–2 from CLRS
11. (858 only) Part b of problem 6–1 from CLRS
12. What suggestions do you have for improving this assignment in the future?

**Submission**

Please make sure that your code (as submitted) compiles on agate with the makefile that you supply. Please make sure that your code runs with the harness because this will be used to grade your assignment.
Electronically submit your source code as described in the instructions from the programming tips sheet on the course web page. Use the assignment name 2 or 2-grad, depending on whether you are enrolled in 858.

Hand in a listing of your source code (2 pages per page, as with a2ps -2) directly to the TA in class, along with your written work, which should contain plots showing the performance of your implementation.

**Evaluation**

In addition to correctness, your work will be evaluated on clarity and efficiency.

Tentative breakdown:

2 heap implementation

8 written problems