University of Waterloo
CS 234 - Data Types and Structures
Spring 2014
Assignment 5
Due Friday July 25th at 4:00 pm

Assignment Guidelines

- Clarifications about the assignment will be posted on Piazza in the thread "Assignment 5 [Official]."
- Instructions on how to submit your assignments to MarkUs will also be posted in Piazza. Your programs must work in the linux.student.cs.uwaterloo.ca environment using Python 2.7.3.
- Use A5-coversheet.pdf for the first page of your assignment or format the top of the first page of your assignment in *exactly* the same manner.
- You may lose up to 20% on a question because it is difficult to read or difficult to understand.

Question 1: Comparing Sorts [20 marks]

For this question you will compare the running time of Heap Sort and Insertion Sort to the default sort provided with Python (i.e. Timsort). In the assignments section of Learn, I have created a file called A5Q1Compare.py. This file imports heapSort and insertionSort from a file called A5Q1.py (which you provide). It then tests these sorts for increasingly larger lists and reports back how long it takes to sort them.

The file A5Q1Compare.py uses three functions that you may not have seen before. The function clock() is used to get the time just before and just after a sorting routine is called in order to measure how long it takes to sort the list. In order to generate a list of random integers you must first call seed() once giving it an integer as input to initialized the random number generator. After calling seed() once, each time you call randint(0, MAX_INT) you get a different random integer between 0 and MAX_INT.

a) In a file called A5Q1.py create a function called heapSort(aList) which given aList will return the list sorted in *descending order* (a slight modification to what is presented in lecture slides 232-237) You must use the in-place version of heapSort *using a min-heap*. In addition to this function, create two helper functions, _siftUp and _siftDown that help you insert and remove items from the heap while maintaining the min-heap property (i.e. that the parent has a lower value than the child).

b) In the same file, A5Q1.py, create a function called insertionSort(aList) that uses the list functions insert() and pop() to implement insertion sort (as presented on lecture slide 243) As in part a) the list must be sorted in *descending order*.

c) In a file called A5Q1Test.py create a function called testHeapSort(aSeed) that tests heapsort in order to ensure that it is sorting lists properly. Here aSeed in an integer value. Do this testing by generating lists of random integers and comparing the results from your implementation of heapsort with the answer that python's default sort provides. To sort aList in descending order in Python use the command aList.sort(reverse=True). You may use the ideas from the file A5Q1Compare.py in order to implement testHeapSort().

d) On paper, compare the time complexity of the implementation you did of Insertion Sort using Python lists that you did in part b) with an implementation that would used linked lists. Which version do you think would typically run faster? Justify your answer.

Your mark will be based on performance in test cases [5 marks] as well inspection of your source code with regard to clarity (i.e. well documented class and function headers, identified preconditions and postconditions, meaningful error message and variable names) as well as organization, simplicity and efficiency. [12 marks] The written portion d) is worth 3 marks.

Question 1 Deliverables:

- On MarkUs, submit a file called A5Q1.py which contains the functions heapSort(), _siftUp(), _siftDown() and insertionSort().
- On MarkUs, submit a file called A5Q1Test.py which contains the function testHeapSort().
- In the assignment drop boxes on the 4th floor of the MC, submit the answer to 1 d)

Question 2: Comparing Sorts[8 marks]

On paper show where the following values would end up if you inserted them in exactly this order, in a heap, using the hash function h(*key*) = *key* mod 13. The values are 19, 110, 32, 71. Show your calculations.

a) using linear probing (i.e. step size of 1)

b) using double hashing where the second hash function is hp(*key*) = 1 + *key* mod 9