CSC 228-01 Data Structures and Algorithms, Spring 2018
Instructor: Dr. Natarajan Meghanathan

Quiz 2 (Take Home): Dynamic Array Implementation of the List ADT: Doubling the List Size vs. Incrementing the List Size by One: *Memory and Run Time Analysis*

Submission: Submit a hardcopy/printout of the report in class): Due: Feb. 9th @ 1 PM

We saw in the dynamic array implementation of the List ADT that for frequent insertion, increasing the size of the List by 1 (one) memory unit is more likely not a better option than doubling the size of the List, whenever needed. In this quiz, we will verify whether this hypothesis is true from the perspectives of both memory usage and actual run time.

You are given the code for the dynamic array implementation of the List ADT. The main function has the timers setup to measure the time it takes to insert a sequence of integers at the end of the List.

Doubling the List: The current implementation of the *insertAtIndex* function in the code given to you doubles the size of the List whenever needed (i.e., calls the *resize* function with a value 2**maxSize*). You could run the code as it is and measure the average time per integer insertion (in nanoseconds, as setup in the main function) for list size values of 1000, 2000, 3000, ..., 10000 (in increments of 1000). For each list size, you will run 50 trials and average the time per insertion. For all list sizes and trials, the range of integers is [1...5000]; that is, *maxValue* is 5000.

Incrementing the size of the List by One: Modify the *insertAtIndex* function code such that it calls the resize function with a value maxSize + 1. After the modification, run the code and measure the average time per integer insertion (as is done for doubling the List) for list size values ranging from 1000 to 10000, in increments of 1000. For all the list size values, the number of trials is 50 and the range of integers is [1...5000].

You could notice that your program stops with a *bad_alloc* error message for a certain value of list size and larger values henceforth (say, for list size of 6000 and larger). In that case, do some online research (for bad_alloc error message in C++) and reason out why you got the error message.

Now, continue running the program for larger list size values even if you get the bad_alloc error message. Note down the trial number at which the bad_alloc error message appears and your program stops. You could notice that the number of trials your program ran successfully and then stopped (due to the bad_alloc problem) decreases with increase in the list size values.

Report:

(1 - 24 pts) The entire code (the List class and the main function) with the *insertAtIndex* function of the List class modified to increment the size of the array by one whenever resizing is needed.

(2 - 8 pts) Screenshots of the output for list size values of 1000 and 10000 in the case of doubling the List.

(3 - 8 pts) Screenshots of the output for list size value of 1000 and the list size value for which you start getting the bad_alloc error message.

(4 - 20 pts) Tabulate the average run time (in nanoseconds) observed for list size values of 1000, ..., 10000 (in increments 1000) for the cases of doubling the list size and incrementing the list size by one (in the latter case, tabulate the run times for list size values for which the program runs successfully for all trials without the bad_alloc problem). **Interpret your results** (explain why they are significantly different or approximately the same for both the strategies, depending on what you observe).

(5 - 20 pts) Give a detailed **explanation** of why you think the bad_alloc error message appeared when you increment the list size by one and not when you doubled the list size in the experimental runs conducted.

(6 - 20 pts) In the case of incrementing the list size by one, tabulate the number of successful trials of your program for each value of list size, ranging from 1000, ..., 10000 (in increments of 1000). **Explain** why the number of successful trials of your program (after the bad_alloc problem starts appearing) decreases with increase in the list size?