

CSC 228-01 Data Structures and Algorithms, Fall 2019
Instructor: Dr. Natarajan Meghanathan

Project 1: Dynamic Array Implementation of the List ADT: Doubling the List Size vs. Incrementing the List Size by One: Memory and Run Time Analysis

Due by: Sept. 12th, 11.59 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 project, we will verify whether this hypothesis is true from the perspectives of both memory usage and actual run time.

You are given the code (posted in Canvas) 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 (Case i)

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

Case ii: 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 Case i) 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.

Case iii: Enhance the given code such that the program will run (while incrementing the list size by 1) for all the 50 trials without stopping with a *bad_alloc* error message. Collect the run times (in nano seconds) for list size values of 1000 to 10000 in increments of 1000; the range of the integers is [1...5000].

Submission:

(1-10 pts) Code for Case-ii as a separate file

(2-20 pts) Code for Case-iii as a separate file

Submit your responses for the following items 3-8 put together as a Single PDF file

(3 - 5 pts) Screenshots of the output for Case i with list size values of 1000 and 10000.

(4 - 5 pts) Screenshots of the output for Case ii for list size value of 1000 and the list size value for which you start getting the *bad_alloc* error message.

(5 - 5 pts) Screenshots of the output for Case iii with list size values of 1000 and 10000.

(6 - 20 pts) Give a detailed **explanation** of why you think the *bad_alloc* error message appeared for Case ii and not under Cases i and iii.

(7 - 15 pts) In the case of Case ii, 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?

(8 - 20 pts) Tabulate (one column for each case) the average run time (in nanoseconds) observed for list size values of 1000, ..., 10000 (in increments 1000) for each of the Cases i, ii and iii. **Interpret your results** (explain why they are significantly different or approximately the same for all the three cases, depending on what you observe).