CS 302 – Assignment #01

Purpose: Refresh concepts regarding C++ simple I/O, functions, object oriented programing, variable scoping, and compilation/linking. Verify installation of development environment.
Due: Tuesday (9/05) → Must be submitted on-line before class.
Points: Part A → 50 pts Part B → 50 pts

Reading/References

Chapter 1, Data Structures and Algorithms

Source: www.xkcd.com/1425

Assignment:

Part A:

In computer science, the longest increasing subsequence[1] problem is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique. For our assignment, we will find the length of the longest increasing subsequence (without return the actual sequence).

For example, given the set: {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15}, the longest increasing subsequence is {0, 2, 6, 9, 11, 15}. This subsequence has length six; the input sequence has no seven-member increasing subsequences. The longest increasing subsequence in this example is not unique: for instance, {0, 4, 6, 9, 11, 15} or {0, 4, 6, 9, 13, 15} are other increasing subsequences of equal length in the same input sequence.

Longest increasing subsequences are studied in the context of various disciplines related to mathematics, including genome sequencing, random matrix theory, representation theory, and physics.

Algorithm 1

The simplest approach is just try every possible combination of increasing elements. This can be done fairly easily via recursion. For each number, there are two possibilities;

- Include current value in LIS if it is greater than the previous element in LIS and recurse for remaining items.
- Exclude current value from LIS and recurse for remaining items.

Finally, we return maximum value obtained by including or excluding current item. The base case of the recursion would be when no items are left.

Algorithm 2

There is a more efficient, but slightly more conceptually complicated algorithm using a dynamic programming[2] approach. Dynamic programming is a method for solving a complex problems by breaking it down into a collection of simpler subproblems, solving each of those subproblems, and storing their solutions.

---

1 For more information, refer to: https://en.wikipedia.org/wiki/Longest_increasing_subsequence
2 For more information, refer to: https://en.wikipedia.org/wiki/Dynamic_programming

For this problem, the recursive solution exhibits overlapping subproblems. If we draw the recursion tree of the solution, we can see that the same sub-problems are getting computed again and again. Problems having optimal substructure and overlapping subproblems can be solved by using Dynamic Programming, in which subproblem solutions are remembered/stored rather than computed again and again.

We can also solve this problem in bottom-up manner. In the bottom-up approach, we solve smaller sub-problems first, then solve larger sub-problems from them. The following bottom-up approach computes ***maxLIS***[***i***], for each > **0** <= ***i*** < ***length***, which stores the length of the longest increasing subsequence (LIS) of sub-array ***arr***[**0**..***i***] that ends with arr[i]. To calculate ***maxLIS***[***i***], we consider LIS of all smaller values of ***i*** (say ***j***) already computed and pick the maximum ***maxLIS***[***j***] where ***arr***[***j***] is less than the current element ***arr***[***i***].

The complete algorithm is as follows:

```
verify length ≠ 0 and arr ≠ NULL
create new array, maxLIS of length size

for(int i=0; i<length; i++)
   maxLIS[i] = 1;
   for(int j=0; j < i; j++)
           if(arr[i]>arr[j])
                   maxLIS[i] = max(maxLIS[i], maxLIS[j]+1);

int result = 0;
for(int i=0; i<length; i++)
   if(maxLIS[i]>result)
           result = maxLIS[i];

return  result;
```

For a more complete explanation, refer to: https://www.youtube.com/watch?v=4fQJGoeW5VE

Data Structure

In order to store the numbers efficiently, we will use a dynamically allocated array. The size of the array is provided via the command line.

Class Descriptions

- Longest Increasing Subsequence, **lis**, Class
  The longest increasing subsequence class will implement both algorithms and some support functions. A header file and implementation file will be required.

lis
-length: int
-*array: int
-LIMIT=7000: static const int
-MINLEN=10: static const int

-MAXLEN=100000: static const int
+lis()
+~lis()
+displayList(): void
+makeList(int): bool
+lisBF(): int
+listDY(): int
+lisREC(int, int): int

Function Descriptions

- The *lis()* constructor function will initialize class variables as appropriate.
- The *~lis()* destructor function should free the dynamically allocated memory.
- The *dissplayList()* function should display the numbers in the array, ten per line (width of 7). Refer to the example for output formatting.
- The *makeList()* function should dynamically create an array, of the passed size, and populate the array with random numbers. The length must be between MINLEN and MAXLEN (inclusive). The random numbers should be generated by rand % LIMIT.
- The *lisBF()* function should determine the longest increasing subsequence using a brute force method by calling the *lisREC()* function.
- The *lisREC()* function should recursively determine the longest increasing subsequence. The arguments are the current index and the previous element.
- The *lisDY()* function should determine the longest increasing using a dynamic programming approach (see explanation).

You may any additional private functions if needed.

Part B:

When completed, use the provided script file to execute the program on a series of different array sizes. The script will write the execution times to a text file.

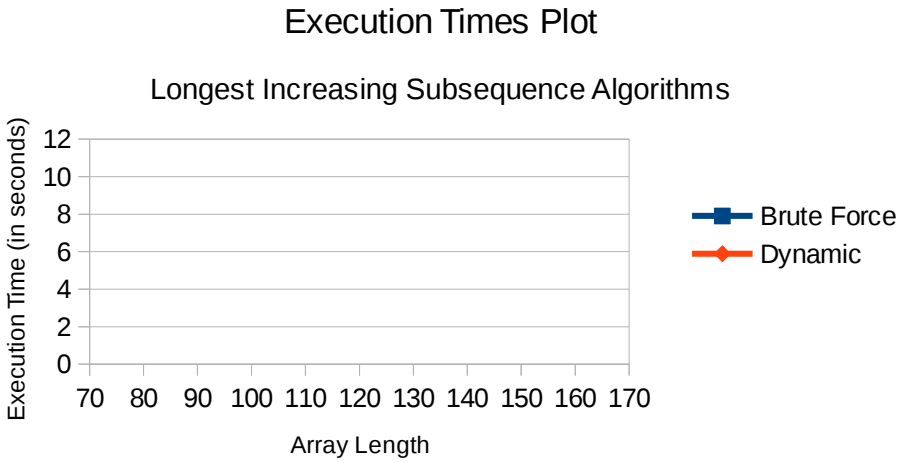
Enter the execution times into a spreadsheet (based on the array lengths) and create a line chart plot of the execution times vs array length for each algorithm. The results are provided in minutes and seconds format and for clarity, however only seconds will be entered in the spreadsheet. Refer to the example for how the plot should look.

Create and submit a write-up with an explanation not to exceed ~500 words including the following:

- Name, Assignment, Section
- Description of the machine used for obtaining the execution times (processor, RAM).
- Copy of the chart (cut-and-pasted from spreadsheet).
- Explanation of the results, comparing the algorithms
  - some comments about why the executions times were similar or were different.

Note, due to different hardware, execution times for each submittal will be different (very different). You should use a word-processor (open document, word format, or Google Docs, etc.) but must submit the file version as a PDF file.

Example Plot:
Below is an example of the execution times plot (excluding the second, algorithm 2, execution times). This incomplete example is to show the appropriate format.



The final chart should be complete and show the times for both algorithms (instead of just one as shown in the example above.

Submission:
When complete, submit:

- Part A → A copy of the **source files** via the class web page (assignment submission link) by class time on or before the due date. The source files, with an appropriate *makefile*, should be placed in a ZIP folder.
- Part B → A copy of the write-up including the chart (see example). Must use PDF format. Other formats will not be accepted (and receive 0 pts).

Assignments received after the due date/time will not be accepted.

You may re-submit as many times as desired. Each new submission will require you to remove (delete) the previous submission. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information.

Reminder: Copying code from someone else or from the net will result in a zero for the assignment and referral to the Office of Student Conduct.

Example Executions:

Below are some sample executions for the program. *Note*, the **ed-vm%** is the prompt.

```
ed-vm%
ed-vm%  ./main -bf 25
************************************************************
CS 302 - Assignment #1
Longest Increasing Subsequence Algorithms.

Algorithm: Brute Force

Length: 25
List:
   4383    886   5777    915   1793   3335   5386    492   4649   5421
   3362   1027   5690   6059   1763   2926   5540   5426   2172   6736
   5211   4368   2567    429   3782
Longest Subsequence = 9
ed-vm%
ed-vm%

ed-vm%
ed-vm% ./main -dy 25
************************************************************
CS 302 - Assignment #1
Longest Increasing Subsequence Algorithms.

Algorithm: Dynamic Programming

Length: 25
List:
   4383    886   5777    915   1793   3335   5386    492   4649   5421
   3362   1027   5690   6059   1763   2926   5540   5426   2172   6736
   5211   4368   2567    429   3782
Longest Subsequence = 9
ed-vm%
ed-vm%
```