Micah Geertson
Justin Stewart
CPSC 335
Project 2
03/25/2016

Programming Assignment #2

Table of Contents

End To Beginning Pseudocode

Input: n, a positive integer, array A of n comparable elements
Output: array R that contains longest non-decreasing subsequence

```
def End_To_Beginning (n, A)
        //Initialize H array to zero
        For i = 0 to n do
                H[i] = 0
        End for

        //Calculate the H values
        For i = n - 2 to 0 do
                For j = i + 1 to n do
                        if A[i] <= A[j] do
                                if H[i] <= H[j] do
                                        H[i] = H[j] + 1
                                End if
                        End if
                End For
        End For

        max = H[0]

        For i = 1 to n do
                if max < H[i]
                        max = H[i]
        End For

        increment max

        //Declare space for R subsequence array
        R[max]

        index = max - 1

        j = 0

        For i = 0 to n do
                if H[i] == index
                        R[j] = A[i]
                        decrement index
                        increment j
                End if
        End For
```

Powerset Pseudocode

Input: n, a positive integer, array A of n comparable elements
Output: array R that contains longest non-decreasing subsequence

```
def Powerset (n, A)
        //Allocate space for bestSet array
        bestSet[n+1]
        bestSize = 0

        //Generate Powerset
        stack[n+1]
        stack[0] = 0
        k = 0

        While true do
                if stack[k] < n do
                        stack[k+1] = stack[k] + 1
                        increment k
                else
                        increment stack[k-1]
                        decrement k
                End if

                if k == 0 do
                        break
                End if

                //Check best set
                if k < 2
                        if k > bestSize
                                For i = 0 to k + 1
                                        bestSet[i] = stack[i]
                                End For
                                bestSize = k
                                return
                        End if
                else
                        For i = 0 to k-1
                                if A[(stack[i+1]-1)] > A[(stack[i+2]-1)]
                                        return
                                End if
                        End For
                End if

                if k > bestSize
                        For I = 0 to k + 1
                                bestSet[i] = stack[i]
                        End For

                        bestSize = k
                        return
                else
                        return
                End if
        End While
```

input: n, a positive integer, Array A of n comparible elements
output: Array R that contains longest non-decreasing subsequence.

```
def End-to-Beg(n, A)
n+1 { H[n] = 0
    for i = n-2 to 0 do
        for j = i+1 to n do
            if A[i] <= A[j] do
                if H[i] <= H[j] do
                    H[i] = H[j] + 1
1 { max = H[0]
2n { for i = 1 to n do
        if max < H[i]
            max = H[i]
4 { increment max
    R[max] → ?
    index = max - 1 → i
    j = 0 → i
4n+1 { for i = 0 to n do
        if H[i] == index
            R[j] = A[i]
            decrement index
            increment j
```

$\frac{n-0}{1} + 1$

$(n+1)4 = 4n+1$.

G = 4n+1

$\frac{3n^2-15n+21}{2}$

A: $\frac{n-0}{1} + 1 = (n+1)*1 = n+1$

B: $\frac{n-(i+1)}{1} + 1 = \frac{n-i-1+1}{1} = n-i$

C: 1 + max(2, 0) = 3

D: 1 + max(1, 0) = 2

E: 2n

F: 1 + max(1, 0) = 2

H: 1 + max(3, 0) = 4

$B = \sum_{i=0}^{n-2} (n-i)*3$

$= 3\sum_{i=0}^{n-2} n - 3\sum_{i=0}^{n-2} i$

$\frac{3(n-2)[(n-2)+1]}{2} - (n-2)3$

$= \frac{3(n-2)[n-1]}{2} - (3n-6)$

$= \frac{3(n^2-3n+3)}{2} - 3n+6$

$= \frac{3n^2-9n+9}{2} - 3n+6$

$= \frac{3n^2-9n+9}{2} - \frac{6n+12}{2}$

$A = \frac{3n^2-15n+21}{2}$

$(n+1) + \frac{3n^2-15n+21}{2} + 1 + 2n + 4 + 4n + 1$

$= n+1 + \frac{3n^2}{2} - \frac{15n}{2} + \frac{21}{2} + 1 + 2n + 4 + 4n + 1$

$= \left(\frac{3n^2}{2}\right) + \left(n - \frac{15n}{2} + 2n + 4n\right) + \left(1 + \frac{21}{2} + 1 + 4 + 1\right)$

$= \frac{3n^2}{2} + -\frac{1}{2}n + 14$ ← total

← Sum

input: n, a positive integer, an Array A of n comparible elements.
output; Array R that contains longest non-decreasing subsequence.

```
def powerset(n, A)
    bestSet[n+1]  ---> 1
    bestSize = 0  ---> 1

    //Generate power set

    stack[n+1]    ---> 1
    stack[0] = 0  ---> 1
    k = 0         ---> 1

    while true do
        if stack[k] < n do
            stack[k+1] = stack[k] + 1
            increment k
        else
            increment stack[k-1]
            decrement k
        if k == 0 do
            break
        //check stack against best set
        if k < 2
            if k > bestSize
                for i = 0 to k+1
                    bestSet[i] = stack[i]
                bestSize = k  ---> 1
                exit          ---> 1
        else
            for i = 0 to k-1
                if A[(stack[i+1]-1)] > A[(stack[i+2]-1)]
                    exit
        if k > bestSize
            for i = 0 to k+1
                bestSet[i] = stack[i]
            bestSize = k  ---> 1
            exit          ---> 1
        else
            exit ---> 1
```

$$5 = 3\sum_{k=0}^{n\cdot 2^n} k + \sum_{k=0}^{n\cdot 2^n} 4$$

$$= 3(n\cdot 2^n) + 4 = (3n\cdot 6^n + 4) + 5$$

$$= 3n\cdot 6^n + 9$$

$O(n\cdot 2^n)$

$1 + \max(2,2) = 3$

+

$1 + \max(1,0) = 2$

$1 + \max(k+4, 0)$ → $= k+5$

$k+1$ → $1 = (k+2)+2$

$1 + \max(k+5, 2k)$

$= 2k+1$

$= 3k+4$

$\underbrace{k-1}_{1} + 1 = k$ → $2k$

$1 + \max(1,0) = 2$

$1 + \max(k+2, 1)$

$k+3$

End To Beginning C++ Code

```
// Assignment 2: Longest non-decreasing subsequence problem, end-to-beginning algorithm

/***************************************
* Name: Micah Geertson & Justin Stewart         *
* CPSC 335-01 13115                             *
* Date: 03/20/2016                              *
***************************************/


// Given a sequence of elements the program finds a subsequence of it in which the subsequence's
// elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.

// The program reads the number of elements in the sequence, then the elements and outputs the
sorted
// sequence and the running time.
// INPUT: a positive integer n and a list of n elements
// OUTPUT: a longest non-decreasing subsequence of the initial sequence

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <chrono>

using namespace std;

void print_sequence(int, float*);
// function to print a sequence, given the number of elements and
// the actual sequence stored as an array


int main() {

  int n, i, j, max, index;
  float *A, *R;
  int *H;


  // display the header
  cout << endl << "CPSC 335-x - Programming Assignment #2" << endl;
  cout << "Longest non-decreasing subsequence problem, end-to-beginning algorithm" << endl;
  cout << "Enter the number of elements in the sequence" << endl;

  // read the number of elements
  cin >> n;
```

```
// allocate space for the input sequence and array H
A = new float[n];
H = new int[n];


// read the sequence
cout << "Enter the elements in the sequence" << endl;

for( i=0; i < n; i++)
{
        cin >> A[i];
}

// print the sequence
cout << "Input sequence" << endl;
print_sequence(n,A);

// Start the chronograph to time the execution of the algorithm
auto start = chrono::high_resolution_clock::now();

// loop to populate the array H with 0 values
for(i=0; i< n; i++)
{
        H[i] = 0;
}


// loop to calculate the values of array H
for ( i = n-2;  i>= 0; i--)
{
        for ( j = i+1; j < n ; j++)
        {
        // WRITE THE CODE THAT IS AN IF CONDITION THAT DECIDES WHETHER
        // TO CHANGE OR NOT THE VALUE OF H[i]
                if(A[i] <= A[j])
                {
                        if(H[i] <= H[j])
                                H[i] = H[j] + 1;
                }
        }
}

// calculate in max the length of the longest subsequence by adding 1
// to the maximum value in H
max = H[0];

for( i=1; i< n; i++)
```

```
        if (max < H[i])
        max = H[i];

max ++;

// allocate space for the subsequence R
R = new float[max];

// add elements to R by whose H's values are in decreasing order, starting
// with max-1
// store in index the H values sought

index = max - 1;

// store in j the index of the element appended to R
j = 0;

for(i=0; i< n; i++)
{
        if (H[i] == index)
        {
        // WRITE THE BLOCK OF STATEMENTS TO ADD A[i] TO THE R SEQUENCE BY
        // STORYING IT INTO R[j], DECREMENTING index AND INCREMENTING j
                R[j] = A[i];
                index--;
                j++;
        }
}

// End the chronograph to time the loop
auto end = chrono::high_resolution_clock::now();

// write the output
cout << "The longest non-decreasing subsequence has length " << endl;
cout << max << endl;
cout << "The longest non-decreasing subsequence is" << endl;
print_sequence(max, R);


// print the elapsed time in seconds and fractions of seconds
int microseconds = chrono::duration_cast<chrono::microseconds>(end - start).count();
double seconds = microseconds / 1E6;
cout << "elapsed time: " << seconds << " seconds" << endl;

// de-allocate the dynamic memory space
delete [] A;
delete [] H;
delete [] R;
```

```
  return EXIT_SUCCESS;
}

void print_sequence(int n, float *seq)
{
// function to print a sequence, given the number of elements and
// the actual sequence stored as an array
// n represents the number of elements in the sequence
// seq represents the actual sequence
// WRITE THE CODE TO PRINT THE ELEMENTS OF A SEQUENCE seq WITH n ELEMENTS

        for (int i = 0; i < n; i++)
        {
                cout << seq[i] << " ";
        }
        cout << endl;
}
```

Powerset C++ Code

```
// Assignment 2: Longest non-decreasing subsequence problem, power set algorithm
/***************************************
* Name: Micah Geertson & Justin Stewart        *
* CPSC 335-01 13115                            *
* Date: 03/20/2016                             *
***************************************/

// Given a sequence of elements the program finds a subsequence of it in which the subsequence's
// elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.

// The program reads the number of elements in the sequence, then the elements and outputs the
sorted
// sequence and the running time.
// INPUT: a positive integer n and a list of n elements
// OUTPUT: a longest non-decreasing subsequence of the initial sequence

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <chrono>

using namespace std;

void print_sequence(int, float*);
// function to print a sequence, given the number of elements the actual sequence stored as an array

void printPowerset(int, int*, int&, float*);
// function to generate the power set of {1, 2, …first argument} and retrieve the best set

void checkSet(int[], int, int*, int&, float*);
// function to check the currently generated set stack of size k against
// the current best set bestSet of size bestSize

int main() {

  int n, bestSize, i;
  float *A, *R;
  int *bestSet;

  // display the header
  cout << endl << "CPSC 335-x - Programming Assignment #2" << endl;
  cout << "Longest non-decreasing subsequence problem, powerset algorithm" << endl;
  cout << "Enter the number of elements in the sequence" << endl;

  // read the number of elements
```

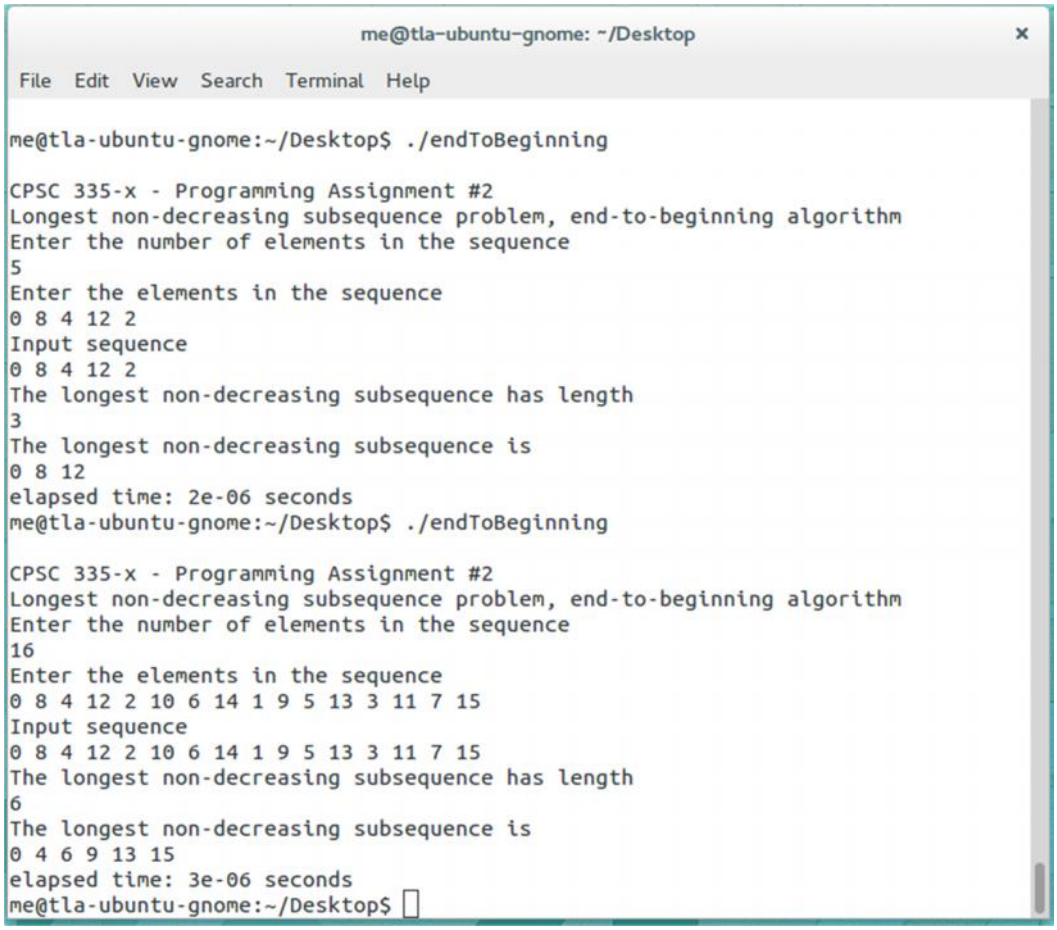
```
cin >> n;

// allocate space for the input sequence and array R
A = new float[n];
R = new float[n];

// read the sequence
cout << "Enter the elements in the sequence" << endl;

for( i=0; i < n; i++)
        cin >> A[i];

// print the sequence
cout << "Input sequence" << endl;
print_sequence(n,A);

// Start the chronograph to time the execution of the algorithm
auto start = chrono::high_resolution_clock::now();

// allocate space for the best set; initial its size is 0
bestSet = new int[n+1];
bestSize = 0;

// calculate the best sequence
printPowerset(n, bestSet, bestSize, A);


// retrieve the indices for generating the subsequence
for(i=0;i<bestSize;i++)
        {// decrease each index by one since the indices of array A are in
        // the range 0..n-1 and not 1..n
        R[i]=A[bestSet[i+1]-1];

}
// End the chronograph to time the loop
auto end = chrono::high_resolution_clock::now();

// display the output
cout << "The longest non-decreasing subsequence has length " << endl;
cout << bestSize << endl;
cout << "The longest non-decreasing subsequence is" << endl;
print_sequence(bestSize, R);

// print the elapsed time in seconds and fractions of seconds
int microseconds =  chrono::duration_cast<chrono::microseconds>(end - start).count();
double seconds = microseconds / 1E6;
cout << "elapsed time: " << seconds << " seconds" << endl;
```

```
 // de-allocate the dynamic memory space
 delete []A;
 delete []R;
 return EXIT_SUCCESS;
}

void print_sequence(int n, float *seq)
{
// function to print a sequence, given the number of elements the actual sequence stored as an array
// n represents the number of elements in the sequence
// seq represents the actual sequence
// WRITE THE CODE TO PRINT THE ELEMENTS OF A SEQUENCE seq WITH n ELEMENTS
        for (int i = 0; i < n; i++)
        {
                cout << seq[i] << " ";
        }
        cout << endl;
}

void printPowerset (int n, int *bestSet, int &bestSize, float *A)
{
// function to generate the power set of {1, .., n} and retrieve the best set
// n represents the maximum value in the set
// bestSet represents the set
// bestSize is the size of the bestSet

 int *stack,k;

// allocate space for the set
 stack = new int[n+1];
 stack[0]=0; /* 0 is not considered as part of the set */
 k = 0;

 while(1){
        if (stack[k]<n)
        {
                stack[k+1] = stack[k] + 1;
                k++;
        }
   else
        {
                stack[k-1]++;
                k--;
        }

        if (k==0)
        break;
```

```
        checkSet(stack, k, bestSet, bestSize, A);
 }

 // deallocate space for the set
 delete [ ] stack;
 return;
}

void checkSet(int *stack, int k, int *bestSet, int &bestSize, float *A)
// function to check the currently generated set stack of size k against the current
// best set bestSet of size bestSize
{
 int i;

 // check that the indices in stack generate a subsequence of non-decreasing order
 if (k < 2)
 {
        // the set contains a single index so the subsequence is in non-decreasing order
        if (k > bestSize)
        {
                // we found a better set
                // WRITE CODE TO STORE stack into bestSet and UPDATE bestSize TO k
                for (i = 0; i < k+1; i++)
                {
                        bestSet[i] = stack[i];
                }
                bestSize = k;
                return;
        }
 }
 else
 {
        // the set contains more than a single index so check that the subsequence is in order
        for(i=0;i<k-1;i++)
        {
          // decrease each index by one since the indices of array A are in
          // the range 0..n-1 and not 1..n
        // WRITE CODE (AN IF STATEMENT) TO CHECK THAT THE ELEMENTS IN ARRAY
                // A AT INDICES stack[i+1]-1 AND stack[i+2]-1 ARE IN NON-DECREASING ORDER
                // IF THE TWO ELEMENTS ARE OUT OF ORDER THEN return
                if(A[(stack[i+1]-1)] > A[(stack[i+2]-1)])
                        return;
  }
 }

 // we have an non-decreasing so we compare it against the current best set
 if (k > bestSize)
 {
```

```
        // we found a better set
        // WRITE CODE TO STORE stack into bestSet and UPDATE bestSize TO k

          for (i = 0; i < k+1; i++)
                {
                        bestSet[i] = stack[i];
                }
                bestSize = k;
                return;
 }
 else
        return;
}
```

End To Beginning Output



Powerset Output