Assignment 1: Introduction to Systems Programming

Francis Vo

October 8, 2012

1 Mathematical Analysis

1.1 Algorithm 1

Data: Integer array A of size N
Result: Greatest Sum of Subarray

```
1  for i ← 0 to N do
2  |  for j ← i to N do
3  |  |  s ← 0
4  |  |  for k ← i to j do
5  |  |  |  s ← s + A[k]
6  |  |  end
7  |  |  if s > max then
8  |  |  |  max ← s
9  |  |  end
10 |  end
11 end
```

Algorithm 1: Pseudocode for Basic Enumeration

1.2 Algorithm 2

Data: Integer array A of size N
Result: Greatest Sum of Subarray

```
1 for i ← 0 to N do
2 |  s ← 0
3 |  for j ← i to N do
4 |  |  s ← A[j]
5 |  |  if s > max then
6 |  |  |  max ← s
7 |  |  end
8 |  end
9 end
```

Algorithm 2: Pseudocode for Better Enumeration

6 Code

6.1 Algorithm 1

```cpp
/*
 * Enumeration
 * Loop over each pair of indices i; j and compute the sum from k=i
     to j of a[k].
 * Keep the best sum you have found so far.
 */

using namespace std;


int MaxSubarray(int a[], int n){

  int i,j,k;
  int max = a[0];
  int sum;
  for(i = 0; i < n; ++i ){
    for (j = i; j < n; ++j){
      sum = 0;
      for (k = i; k <=j; ++k){
        sum += a[k];
      }
      if(max < sum){
        max = sum;
      }
    }
  }
  return max;
}
```

alg1.cpp

6.2 Algorithm 2

```
/*
 * Better Enumeration
 * Notice that in the previous algorithm, the same sum is computed
     many times.
 * In particular, notice that sum from k=i to j of a[k] can be
     computed from sum from k=i to j - 1 of a[k] in O(1) time,
     rather than starting from scratch.
 * Write a new version of the frst algorithm that takes advantage
     of this observation.
 */

using namespace std;


int MaxSubarray(int a[], int n){

  int i,j,k;
  int max = a[0];
  int sum;
  for(i = 0; i < n; ++i ){
    sum = 0;
    for (j = i; j < n; ++j){
      sum += a[j];
      if(max < sum){
        max = sum;
      }
    }
  }
  return max;
}
```

alg2.cpp

6.3 Algorithm 3

```
/*
 * Divide and Conquer
 * If we split the array into two halves, we know that the maximum subarray will either be
 *      * contained entirely in the frst half,
 *      * contained entirely in the second half, or
 *      * made of a suffix of the frst half of maximum sum and a prefix of the second half of maximum sum
 * The frst two cases can be found recursively. The last case can be found in linear time.
 */
```

alg3.cpp