Project 1

CS325 — Spring 2015

by Group 2
Vedanth Narayanan
Jonathan Merrill
Tracie Lee

April 25, 2015

1 Theoretical Run-time Analysis

1.1 Algorithm 1

```
maxSubarray(a[1,...,n])
    for each pair(i,j) with 1<=i<=j<=n
        compute a[i]+a[j+1]+...+a[j-1]+a[j]
        keep max sum found so far
return max sum found
```

Asymptotic Analysis
We have $O(n^2)$ pairs * $O(n)$ time to compute each sum $= O(n^3)$.

1.2 Algorithm 2

```
maxSubarray(a[1,...,n])
    for i = 1, ...., n
        sum = 0
        for = i, ...., n
            sum = sum + a[j]
            keep max sum found so far
    return max sum found
```

Asymptotic Analysis
We have $O(n)$ i-iterations (outer loop) * $O(n)$ j-iterations (inner loop) * $O(n)$ for the time to update $= O(n^2)$.

1.3 Algorithm 3

```
%% ENTER PSEUDO-CODE HERE %%
```

Asymptotic Analysis
We have $T(n) = 2T(\frac{n}{2}) + \Theta(n)$. This falls within Case 2 of the Master Method, and therefore yields a solution of $\Theta(nlgn)$.

1.4 Algorithm 4

```
maybeStart = 0
start = 0
end = 0
i = testArray[0]
```

```
sum = testArray[0]
small = Alg4Helper(0,i)
(helper function to determine the minimum between a pair of values)

for j in range(1,len(testArray)):
    i = i + testArray[j]
    if (i - small) > sum:
        start = maybeStart
        end = j+1
        sum = (i - small)
    if i < small:
        maybeStart = j+1
        small = i
return (sum, testArray, testArray[start:end])
```

Asymptotic Analysis
We have $O(n)$ things to compute, therefore this takes $O(n)$ time.

2 Proof of Correctness: Algorithm 3

Base Case

We pass in either an empty array or an array consisting of 1 element. In the first case, an empty array is returned and in the second case the algorithm returns the same array that had been passed in since it is the max subarray within that array.

Inductive Hypothesis

Assume that algorithm 3 correctly returns a max subarray from an array of n+1 elements.

Inductive Step

Termination

3 Testing

4 Experimental Analysis

5 Extrapolation and interpretation