Project 1

CS325 — Spring 2015

---

by Group 2
Vedanth Narayanan
Jonathan Merrill
Tracie Lee

April 26, 2015

1 Theoretical Run-time Analysis

1.1 Algorithm 1

```
maxSubarray(a[1,...,n])
    max = a[0]
    for i = [0...n]
        for j = [i,n]
            sum = 0
            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

```
maxSubarray(a[1,...,n], initial array length)
    length = len(a)

    if length > 1:
        left = left half of array
        right = right half of array
        first = maxSubarray(left, 0)
```

```
        last = maxSubarray(right, 0)
        reverse left
        center = helper(left) + helper(right)
    else:
        first = last = center = a[0]

    if initial array length == len(a):
        PrintResults(max([first, last, center]), a, [first, last, center])

    return max([first, last, center])

helper(a):
    max = a[0]
    sum = 0
    for i in range(0, len(a)):
        sum += a[i]
        if sum > max:
            max = sum
    return max
```

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

```
maxSubarray(a[1,...,n])

maybeStart = 0
start = 0
end = 0
i = a[0]
sum = a[0]
small = minimum of (0, i)

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

```
            end = j+1
            sum = (i - small)
        if i < small:
            maybeStart = j+1
            small = i
    return (sum, a, a[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 maximum contiguous sum of elements S from an array A of $n > 1$ elements.

Inductive Step

If we split A into 2 separate arrays, let L represent the new array from the left side and let R represent the new array from the right side. Then let $s_{i...j}$ represent any sequence of numbers with the largest sum that lies with in S. If the sum of $s_{i...j} = x$, then $max(first, last, center) \leq x$. There are then 3 possibilities:

1. $s_{i...j}$ lies completely within L. In this case, it would follow that the max subarray of L is equal to x which means that $max(first, last, center) \geq first = x$. Because of this, we know that the answer returned is exactly x.

2. $s_{i...j}$ lies completely within R. In this case, it would follow that the max subarray of R is equal to x which means that $max(first, last, center) \geq last = x$. Because of this, we know that the answer returned is exactly x.

3. If $s_{i...j}$ does not lie completely within L or R, then it must start in L and end in R.

3 Testing

4 Experimental Analysis

5 Extrapolation and interpretation