COMP 372 Assignment 1

Mark Griffith

September 19, 2017

1 1.1-2

Similarities:

1. Both are optimization problems (optimal route)
2. Both are weighted graph problems

Differences:

1. TSP is an NP-complete problem whereas shortest-path is a known polynomial problem
2. The solution to TSP is a cycle whereas the solution to the shortest-path is not
3. TSP will visit at least one node twice (first and last node) whereas the shortest-path will never visit a node more than once

2 1.2-2

For insertion sort to beat merge sort, its runtime must be less than that of merge sorts. Therefore we can compare the algorithms runtimes as

$$8n^2 = 64nlg(n)$$

$$n = 8lg(n)$$

By solving the above equation (I used the method of graphing) and noting that the sort does not apply for any n < 1, we see that for **n < 44, insertion sort beats merge sort.**

3 2.1-3

Assuming we are 1 indexing...

```
LINEAR-SEARCH(v, A):
  for i = 1 to A.length:
    if v == A[i]:
      return i
  return NIL
```

Proof

Loop Invariant: at the start of each for loop, $A[i-1]$!= v
Initialization: Prior to the initial for loop, $i = 1$ and $i - 1 = 0$ so $A[i-1] = A[0]$. $A[0]$ cannot equal v for any v because A has no 0th element.
Maintenance: Suppose we are on iteration j of the for loop and that our loop invariant is true for that iteration i.e. $A[j]! = v$.
Termination: The condition that causes the for loop to terminate is $i > A.length =$ n **or** $A[i] = v$. Therefore,

4 2.2-3

Consider linear search again (see Exercise 2.1-3). How many elements of the in- put sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array? How about in the worst case? What are the average-case and worst-case running times of linear search in θ-notation? Justify your answers.

Suppose we are searching for element x in the array. Assuming that each element in the array is equally likely to be x, there is a $\frac{1}{n}$ (where n is the length of the array) chance of x being any given element of the array.

On average, x will be at the middle of the array. Therefore, the average number of elements of the array needed to be checked for x is $\frac{n}{7}2$.

In the worst case scenario (x is not in the array), all the elements of the array must be checked (n elements).

Both cases have a run time of order an. Therefore, both cases have run time $\theta(n)$.

5 2.3-5

Referring back to the searching problem (see Exercise 2.1-3), observe that if the sequence A is sorted, we can check the midpoint of the sequence against and eliminate half of the sequence from further consideration. The binary search algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time. Write pseudocode, either iterative or recursive, for

binary search. Argue that the worst-case running time of binary search is O(lg n).

```
min = A[0]
max = A[A.length - 1]
found = False

while (min <= max)
  if value = A[mid] then
    found = True
  else if value < A[mid] then
    max = mid - 1
  else
    min = mid + 1
  fi
done
```

For the above algorithm, the worst case scenario is the case in which the value being searched for is either at the min or max element of the array A. In that scenario, the runtime would be θ(lg n) where n is the length of the array. This is because we start with n elements in the array and for each iteration of the loop we halve number of elements. We do this until there only a single value left in the array. This can be represented as

$$n \cdot (1/2)^x = 1$$

where x is the number of times we need to iterate over the loop to find the value. To solve for x, we can rearrange this to be

$$\begin{aligned} n/2^x &= 1 \\ n &= 2^x \\ lg(n) &= x \end{aligned}$$

And so we see that the worst-case runtime of the binary search is T(n) = θ(lg n).

6 3.1-1

To prove that max(f(n), g(n)) = θ(f(n) + g(n)) we must find two constants c_1 and c_2 such that

$$\begin{aligned} &\theta(f(n)) = f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that} \\ &\qquad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0, \max(f(n)) = c_2 g(n) \end{aligned}$$

We know that there must exist an n_0 such that $f(n) \geq 0$ and $g(n) \geq 0$ for all $n \geq n_0$. So for any $n \geq n_0$, $f(n) + g(n) \geq f(n) \geq 0$ and $f(n) + g(n) \geq g(n) \geq 0$. Combining these inequalities we see that $f(n) + g(n) \geq max(f(n), g(n))$ (for $n \geq n_0$. We can rewrite this as $max(f(n), g(n)) \leq c(f(n) + g(n))$ with $c = 1$. Therefore, $max(f(n), g(n)) = O(f(n) + g(n))$ with $c_1 = 1$

Now we can also write $max(f(n), g(n)) \geq f(n)$ and $max(f(n), g(n)) \geq g(n)$ for all $n \geq n_0$. Combining these inequalities we see that $max(f(n), g(n)) \geq \frac{1}{2}(f(n) + g(n))$ for all $n \geq n_0$
Therefore, $\max(f(n), g(n)) = \Omega(f(n) + g(n))$ with $c_2 = \frac{1}{2}$

We can now see that $\max(f(n), g(n)) = \Omega(f(n) + g(n))$ bounded by $c_1 = 1$ and $c_2 = \frac{1}{2}$

7 3-1

8 4.1-2

BRUTE FORCE: find every subarray for each element in the array and keep track of the maximum subarray

```
MaxSubarray(array):
  total = 0
  max = 0
  for i from 0 to array.length - 1:
    total = 0
    for y from i to array.length - 1:
      total += array[y]
      if total > max:
        max = total
  return max
```

9 4.2-1

Use Strassen's algorithm to compute the matrix product (1 3) (6 8) (7 5) (4 2) Show your work

$$
\begin{aligned}
S1 &= 8 - 2 = 6 \\
S2 &= 1 + 3 = 4 \\
S3 &= 7 + 5 = 12 \\
S4 &= 4 - 6 = -2 \\
S5 &= 1 + 5 = 6 \\
S6 &= 6 + 2 = 8 \\
S7 &= 3 - 5 = -2 \\
S8 &= 4 - 2 = 2 \\
S9 &= 1 - 7 = -6 \\
S10 &= 6 + 8 = 14
\end{aligned}
$$

$$
\begin{aligned}
P1 &= 1 * 6 = 6 \\
P2 &= 4 * 2 = 8 \\
P3 &= 12 * 6 = 72 \\
P4 &= 5 * -2 = -10 \\
P5 &= 6 * 8 = 48 \\
P6 &= -2 * 2 = -4 \\
P7 &= -6 * 14 = -84
\end{aligned}
$$

$$
\begin{aligned}
C11 &= P5 + P4 - P2 + P6 = 48 - 10 - 8 - 4 = 26 \\
C12 &= P1 + P2 = 6 + 8 = 14 \\
C21 &= P3 + P4 = 72 - 10 = 62 \\
C22 &= P5 + P1 - P3 - P7 = 48 + 6 - 72 + 84 = 66
\end{aligned}
$$

Therefore, according to Strassen's algorithm, the resulting matrix is

$$
\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} 26 & 14 \\ 62 & 66 \end{bmatrix}
$$

10 4.3-2

To prove $T(n) = O(lgn)$ we must first prove that $T(n) \leq clg(n)$ for $c > 0$. By use of substitution:

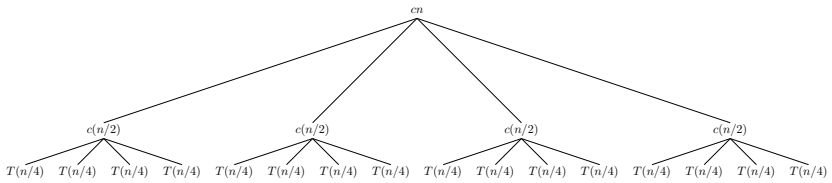
$$
\begin{aligned}
T(n) &\leq clg([n/2]) + 1 \\
&< clg(\frac{n}{2} + 1) + 1 \\
&= clg(n) - clg(2) + 1 \\
&= clg(n) - c + 1 \\
&= clg(n) \qquad\qquad \text{if } c \geq 1
\end{aligned}
$$

Assuming that $T(1) = 1$ we can prove that $T(2) \leq clg(2)$ for $c \geq 1$ as the base case. $T(2) = T(1) + 1 = 2$. Therefore $T(2) \leq clg(2)$ for $c \geq 2$

11 4.4-7

Draw the recursion tree for $T(n) = 4T[n/2] + cn$ where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substi- tution method

Recursion tree for $T(n) = 4T[n/2] + cn$



Cost of the tree:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n} 4^i \cdot c\frac{n}{2^i} \\
&= cn \cdot \sum_{i=0}^{\lg n} 2^i \\
&= cn \cdot \frac{2^{\lg n+1} - 1}{2-1} \\
&= cn \cdot (2 \cdot 2^{\lg n} - 1 \\
&= cn \cdot 2 \cdot 2n - cn \\
&= 2cn^2 - cn
\end{aligned}
$$

$$T(n) = 2cn^2 - cn$$

Assymptotic Bounds

for all n $\geq$ 2 ...
upper:

$$
\begin{aligned}
T(n) &= cn^2 + cn^2 - cn \\
T(n) &\leq 2cn^2 \\
T(n) &= O(n^2)
\end{aligned}
$$

lower:

$$
\begin{aligned}
T(n) &= cn^2 + cn^2 - cn \\
T(n) &\geq cn^2 \\
T(n) &= O(n^2)
\end{aligned}
$$

12 4.5-3