COP 3502 Study Group Sheet: Algorithm Analysis and Linked Lists Solutions

Directions: These are questions from an old exam. Based on the quiz 2 results, not only will it be good for many to practice the current course material, but it would be very useful to go back and get extra practice with recursion and linked list. These questions provide practice on both of these topics together.

1) An O(n^2) algorithm takes 2 ms. to run on an input size of n = 50. On another input size, the algorithm took 72 ms. to run, what was that input size?

Solution

Let T(n) be the run time of the algorithm. Then T(n) = cn^2, for some constant c. Using the given information, we have: T(50) = $c(50)^2$ = 2 ms, thus c = $\frac{1}{1250}ms$. Let x be the input size for which the algorithm takes 72 ms. This gives us the equation:

$$T(x) = \frac{1ms}{1250}x^2 = 72ms$$
$$x^2 = 72 \times 1250$$
$$x^2 = 72 \times 2 \times 625$$
$$x^2 = 144 \times 625$$
$$x^2 = 12^2 \times 25^2$$
$$x = 12 \times 25 = \mathbf{300}$$

2) Write a function that takes in a pointer to the front of a linked list and an integer threshold and returns the number of values in the linked list greater than threshold.

```
struct node {
    int data;
    struct node* next;
};
```

Solution

```
int countBigger(struct node* front, int threshold) {

    int res = 0;
    while (front != NULL) {
        if (front->data > threshold) res++;
        front = front->next;
    }
    return res;
}
```

3) What is the run-time (Big-Oh) in terms of n of the following code segments? (Assume all variables have been previously declared as integers.)

a)
```
int i, j, sum = 0;
for (i=0; i<n; i++) {
  for (j=0; j<i; j++) {
    if (j > i)
      sum++;
  }
}
```

Solution

The inner for loop runs i times. Inside of this for loop is O(1) work. i ranges from 0 to n-1, thus the number of times the inner for loop runs is $\sum_{i=0}^{n-1} i = \frac{(n-1)n}{2} = \boldsymbol{O(n^2)}$**.** This is the run time of the code segment.

b)
```
int i, sum = 0, saven = n;
while (n > 0) {
  for (i=0; i<saven; i++)
    sum++;
  n = n/2;
}
```

Solution

The inner loop ALWAYS runs n times, since saven never changes. The outer loop runs for each of these values: n, n/2, n/4, …The last time it runs, n will be 1. Let k equal the number of times the outer loop runs. Then $n/2^k = 1$ and $n = 2^k$ and $k = \log_2 n$, roughly. It follows that the total run time is **O(nlgn).**

c)
```
int sum = 0;
while (n > 0) {
  sum++;
  n = n/3;
}
```

Solution

The values of n, in terms of the original value of n for each loop iteration are n, n/3, n/9, etc. Let k be the number of times the loop runs. Then $n/3^k = 1$ and $n = 3^k$, so $k = \log_3 n$, roughly. It follows that the run time of the code segment is **O(lg n).**

4) Determine an expression in terms of n for the two following sums:

a) $\sum_{i=n+1}^{2n}(3i-1)$

Solution

$$
\begin{aligned}
\sum_{i=n+1}^{2n}(3i-1) &= \left(\sum_{i=n+1}^{2n}3i\right)-\sum_{i=n+1}^{2n}1\\
&= \left(\left(\sum_{i=1}^{2n}3i\right)-\left(\sum_{i=1}^{n}3i\right)\right)-(2n-(n+1)+1)\\
&= \frac{3(2n)(2n+1)}{2}-\frac{3n(n+1)}{2}-(2n-n-1+1)\\
&= \frac{3n}{2}\big(2(2n+1)-(n+1)\big)-n\\
&= \frac{3n}{2}(4n+2-n-1)-n\\
&= \frac{3n}{2}(3n+1)-n\\
&= \frac{9}{2}n^2+\frac{3n}{2}-n\\
&= \boldsymbol{\frac{9}{2}n^2+\frac{n}{2}}
\end{aligned}
$$

b) $\sum_{i=7}^{n}8$

Solution

$$\sum_{i=7}^{n}8 = 8(n-7+1) = 8(n-6) = \boldsymbol{8n-48}$$

5) Use the iteration technique to solve the following recurrence:

$$T(n) = T(n/2) + n, T(1) = 1$$

(Hint: In solving this recurrence utilize the fact that $\sum_{i=0}^{k} \frac{1}{2^i} \approx 2$ for large values of k.)

Solution

So, let's iterate a couple times:

$$\begin{aligned} T(n) &= T(n/2) + n \\ &= T(n/4) + n/2 + n \\ &= T(n/8) + n/4 + n/2 + n \end{aligned}$$

After k steps we have:

$$T(n) = T(n/2^k) + n \sum_{i=0}^{k-1} \frac{1}{2^i}$$

Set k such that $n/2^k = 1$, so $n = 2^k$. We can bound T(n) as follows by relaxing the sum to go to infinity:

$$T(n) \leq T(1) + n \sum_{i=0}^{\infty} \frac{1}{2^i} = 1 + 2n = \mathbf{O(n)}$$

6) Write a recursive function that returns the largest value in a linked list of positive integer values. If the list passed to the function is null, return 0.

```
struct node {
    int data;
    struct node* next;
};
```

Solution

```
int maxVal(struct node* front) {
   if (front == NULL) return 0;
   int rest = maxVal(front->next);
   return rest > front->data ? rest : front->data;
}
```

7) Write a function that returns 1 if a linked list is in ascending sorted order with no duplicates and 0 otherwise. If the list is empty, your function should return 1.

```
struct node {
    int data;
    struct node* next;
};
```

Solution

```
int inOrder(struct node* front) {
    if (front == NULL || front->next == NULL) return 1;
    if (front->data >= front->next->data) return 0;
    return inOrder(front->next);
}
```

Iterative Solution

```
int inOrder(struct node* front) {
    if (front == NULL || front->next == NULL) return 1;

    while (front->next != NULL) {
        if (front->data >= front->next->data) return 0;
        front = front->next;
    }
    return 1;
}
```