Assignment 2

William Jagels

October 5, 2016

1. Integer Powers

1.1. Function

```
def power(x, y):
    y = int(y)
    if y == 1:
        return x
    r = power(x, y/2)
    if y % 2 == 0:
        return r * r
    return x * r * r
```

1.2. Recurrence Equations

Let $n = y$

$$T(n) = T(n/2) + c_2$$

$$T(n) = c_2 \log_2 n + c_1$$

For even case, $c_2 = 1$ because of one multiplication per recursion. For the odd case, $c_2 = 2$ because of the additional multiplication. $c_1 = 0$ because the base case of $T(1)$ requires zero multiplications.

$$T(n) = \begin{cases} \log_2 n, & \text{if n is even} \\ 2\log_2 n, & \text{if n is odd} \end{cases} \tag{1}$$

2. Direct vs Recursive

Recursive solution

$$T(n) = 2T(n/2) + n\log_2 n + \log_2 n$$

$$f(n) = n\log_2 n + \log_2 n = O(n \log n)$$

$$T(n) = n^{\log_2 2} * f(n)$$

$$T(n) = O(n \log_2^2 n)$$

Therefore, recursive solution is better.

3. Flood fill

```
def flood(color, x, y, target, N):
    if x > N or y > N:
        return
    if color[x, y] = BLACK:
        return
    if color[x, y] != target:
        color[x, y] = target
    else:
        return
    flood(color, x + 1, y, target)
    flood(color, x - 1, y, target)
    flood(color, x, y + 1, target)
    flood(color, x, y - 1, target)
```

x and y are the coordinates of an arbitrary point within the region to be filled, and target is the color desired.

4. Index equals element

4.1. Algorithm

```
def indexeq(A):
    return indexeq_h(A, 0, len(A)-1)

def indexeq_h(A, l, u):
    if l == u:
        return A[l] == l
    if u - l == 1:
        return A[l] == l or A[u] == u
    mid = l + int(u/2)
    if A[mid] == mid:
        return True
    if A[mid] > mid:
        return indexeq_h(A, mid, u)
    return indexeq_h(A, l, mid)
```

4.2. Time complexity

$\Theta(\log_2 n)$ because at every step, the search space is halved due to the assumption that the list is sorted.

$$T(N) \leq \begin{cases} N, & if\ N \leq 2 \\ \lg(N), & if\ N > 2 \end{cases} \tag{2}$$

In cases where the middle element of the list satisfies the condition, the runtime will be $T(1)$. At worst, the algorithm will take $T(\log_2 N)$ steps.

5. Graphing

5.1. Pseudocode

```
def makegraph(x, y, r):
    if r == 0:
        return
    DrawSquare(x, y, r)
    n = r\2
    makegraph(x + r, y + r, n) # Q1
    makegraph(x - r, y + r, n) # Q2
    makegraph(x - r, y - r, n) # Q3
    makegraph(x + r, y - r, n) # Q4
    return
```

5.2. Recurrence equation

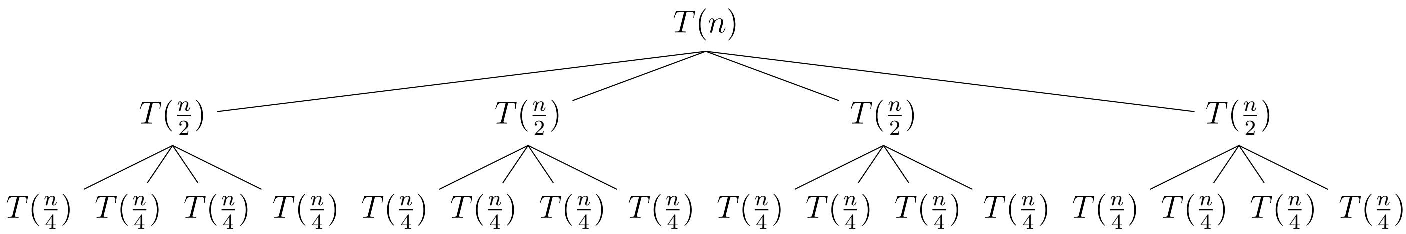
$$T(r) = 4T(r/2) + 1$$

$$T(r) = \frac{1}{12}((3c_1 + 4)r^2 - 4)$$

$c_1 = 4$ because $T(1) = 1$

$$T(r) = \frac{1}{12}((3(4) + 4)r^2 - 4)$$

5.3. Recursion Tree



6. Majority element

```
def freq(A, candidate):
    c = 0
    for e in A:
        if e == candidate:
            c += 1
    return c

def maj(A):
    if len(A) == 1:
        return A[0]
    candidate_l = maj(A[:len(A)//2])
    candidate_r = maj(A[len(A)//2:])
    if candidate_l == candidate_r:
        # Best case
        return candidate_l
    if freq(A, candidate_l) > len(A)//2:
        return candidate_l
    if freq(A, candidate_r) > len(A)//2:
        # Worst case
        return candidate_r
```

7. MinMax

7.1. Recurrence equation

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

Each recursion step performs 2 comparisons, and calls 2 other steps with the left and right halves.

7.2. Recursion tree

$$T(n) + c_2$$

$$T(\tfrac{n}{2}) + c_2 \qquad T(\tfrac{n}{2}) + c_2$$

$$T(\tfrac{n}{4}) + c_2 \quad T(\tfrac{n}{4}) + c_2 \quad T(\tfrac{n}{4}) + c_2 \quad T(\tfrac{n}{4}) + c_2$$

$$T(n) = \frac{c_1 n}{2} + c_2(n-1)$$

$c_2 = 2$ because of the two comparisons per recursive step. Therefore, $c_1 = 4$

8. Master Method

8.1.

$$T(n) = 9T(\frac{n}{3}) + n^2 + 4$$

$$f(n) = n^2 + 4 = O(n^2)$$

$$T(n) = n^{\log_3 9} * f(n)$$

$$h(n) = \frac{n^2 + 4}{n^{\log_3 9}}$$

$$h(n) = \frac{n^2 + 4}{n^2}$$

Same order, so

$$T(n) = O(n^2 \lg n)$$

8.2.

$$T(n) = 6T(\frac{n}{2}) + n^2 - 2$$

$$f(n) = n^2 - 2 = O(n^2)$$

$$T(n) = n^{\log_2 6} * f(n)$$

$\log_2 6 > 2$ so $O(1)$

$$T(n) = O(n^{\log_2 6}) * O(1)$$

$$T(n) = O(n^{\log_2 6})$$

8.3.

$$T(n) = 4T(\frac{n}{2}) + n^3 + 7$$

$$f(n) = n^3 + 7 = O(n^3)$$

$$T(n) = n^{\log_2 4} * f(n)$$

$$T(n) = n^2 * f(n)$$

Lower order than $f(n)$, so

$$T(n) = O(1) * O(n^3)$$

$$T(n) = O(n^3)$$

9. Loop invariant

Proof.

$$A[i] = \sqrt{A[i-1]^2 + X[i]^2}$$

Base case: $n = 1 : A[0] = X[0], A[1] = \sqrt{X[0]^2 + X[1]^2}$
Assume that at ith iteration, $A[i] = \sqrt{A[i-1]^2 + X[i]^2}$
Then, at the $i + 1$th iteration, we have:

$$A[i+1] = \sqrt{A[i]^2 + X[i+1]^2}$$

$$A[i+1] = \sqrt{(\sqrt{A[i-1]^2 + X[i]^2})^2 + X[i+1]^2}$$

$$A[i+1] = \sqrt{A[i-1]^2 + X[i]^2 + X[i+1]^2}$$

□

10. Recurrence equation

$T(n) = 8T(n-1) - 21T(n-2) + 18T(n-3)$ for $n > 2$

$$T(n) = C_1 * r_1^n + C_2 * r_2^n + C_3 * n * r_3^n$$

$$r_1 = 2,\ r_2 = 3,\ r_3 = 3$$

$$T(n) = C_1 * 2^n + C_2 * 3^n + C_3 * n * 3^n$$

$T(0) = 0$ therefore, $C_1 + C_2 = 0$.
$T(1) = 1$ therefore, $C_3 = \frac{C_1}{3} + \frac{1}{3}$.
$T(2) = 2$ therefore, $C_1 = -4$.
Then, $C_2 = 4$ and $C_3 = \frac{-4}{3} + \frac{1}{3} = -1$.

$$T(n) = -4 * 2^n + 4 * 3^n - 1 * n * 3^n$$