University of Nevada, Las Vegas Computer Science 477/677 Spring 2024

Answers to Assignment 4: Due Saturday March 2 2024

1. Write the asymptotic time complexity for each code fragment, using Θ notation.

   (a)
   ```
   for (int i=1; i < n; i++)
     for (int j=i; j > 0; j--)
   ```
   $\Theta(n^2)$

   (b)
   ```
   for (int i=1; i < n; i=2*i)
     for (int j=i; j < n; j++)
   ```
   $\Theta(n)$

   (c)
   ```
   for (int i=1; i < n; i++)
     for (int j=1; j < i; j = j*2)
   ```
   $\Theta(n \log n)$

   (d)
   ```
   for (int i=1; i < n; i++)
     for (int j=i; j < n; j = j*2)
   ```
   $\Theta(n)$

   (e)
   ```
   for (int i=2; i < n; i = i*i)
   ```
   $\Theta(\log \log n)$

   (f)
   ```
   for (int i=1; i*i < n; i++)
   ```
   $\Theta(\sqrt{n})$

2. Give an asymptotic solution to each of these recurrences, using the Bentley-Blostein-Saxe method, otherwise known as the master theorem. Some of them may require substitution.

   (a) $F(n) = 2F(n/2) + n$

   $F(n) = \Theta(n \log n)$

   (b) $F(n) = 4F(n/2) + n^3$

   $F(n) = \Theta(n^3)$

   (c) $F(n) = 4F(n/2) + n^2$

   $F(n) = \Theta(n^2 \log n)$

   (d) $F(n) = 4F(n/2) + n$

   $F(n) = \Theta(n^2)$

   (e) $T(n) = 7\,T(n/7) + n$

   $F(n) = \Theta(n \log n)$

   (f) $T(n) = 9T(n/3) + n^2$

   $T(n) = \Theta(n^2 \log n)$

(g) $T(n) = 8T(n/2) + n^3$

$T(n) = \Theta(n^3 \log n)$

(h) $T(n) = T(\sqrt{n}) + 1$ Use substitution: $m = \log n$.

Let $m = \log n$ and $T(n) = G(m) = G(\log n)$ Then $T(\sqrt{n}) = G(\log \sqrt{n})) = G\left(\frac{\log n}{2}\right) = G(m/2)$. The recurrence becomes $G(m) = G(m/2) + 1$. By the master theorem, $T(n) = G(m) = \Theta(\log m) = \Theta(\log \log n))$

(i) $T(n) = 2T(n-1) + 1$ Use substitution: $n = \log m$, *i.e.* $m = 2^n$.

Let $m = 2^n$, and $G(m) = T(\log m) = T(n)$. Then $T(n-1) = T(\log m - 1) = T(\log(m/2)) = G(m/2)$
The recurrence becomes:
$G(m) = 2G(m/2) + 1$.
By the master theorem, $F(n) = G(m) = \Theta(m) = \Theta(2^n)$

3. Give an asymptotic solution to each of these recurrences using the Akra-Brazzi method, otherwise known as the generalized master theorem.

(a) $F(n) = 2F(n/4) + F(n/2) + 1$

$\alpha_1 = 2, \beta_1 = 1/4, \alpha_2 = 1, \beta = 1/2, \gamma = 0$
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 3$ which is greater than 1. Thus we need to find δ such that $\alpha_1\beta_1^\delta + \alpha_2\beta_2^\delta = 1$. The solution is $\delta = 1$. Thus
$F(n) = \Theta(n^\delta) = \Theta(n)$.

(b) $F(n) = 2F(n/4) + F(n/2) + n$

$\alpha_1 = 2, \beta_1 = 1/4, \alpha_2 = 1, \beta = 1/2, \gamma = 1$
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 1$.
Thus $F(n) = \Theta(n^\gamma \log n) = \Theta(n \log n)$.

(c) $F(n) = 2F(n/4) + F(n/2) + n^2$

$\alpha_1 = 2, \beta_1 = 1/4, \alpha_2 = 1, \beta = 1/2, \gamma = 2$
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 3/8 < 1$.
Thus $F(n) = \Theta(n^\gamma) = \Theta(n^2)$.

(d) $F(n) = F(3n/5) + F(4n/5) + n^2$

$\alpha_1 = 1, \beta_1 = 3/5, \alpha_2 = 1, \beta = 4/5, \gamma = 2$
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = \frac{9}{25} + \frac{16}{25} = 1$.
Thus $F(n) = \Theta(n^\gamma \log n) = \Theta(n^2 \log n)$.

(e) $F(n) = F(n/3) + 4F(2n/3) + 1$

$\alpha_1 = 2, \beta_1 = 1/3, \alpha_2 = 4, \beta = 2/3, \gamma = 0$
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 5 > 1$

Thus we need to find δ such that $\alpha_1\beta_1^\delta + \alpha_2\beta_2^\delta = 1$. The solution is not a nice number. By using binary search, we find $\delta \approx 3.4739$. Thus
$F(n) = \Theta(n^{3.4739})$, but not exactly.

4. Give an asymptotic solution to each these recurrences, using the anti-derivative method.

   (a) $F(n) = F(n - \log n) + \log n$

   $$\frac{F(n) - F(n - \log n)}{\log n} = 1$$

   The left side is asymptotically the derivative of F, and thus we have
   $F'(n) = \Theta(1)$
   $F(n) = \Theta(n)$

   (b) $G(n) = G(n-1) + n^c$ where $c \geq 1$ is a constant.

   $$\frac{G(n) - G(n-1)}{1} = n^c$$

   $G'(n) = \Theta(n^c)$

   $G(n) = \Theta(n^{c+1})$

   (c) $K(n) = K(n - \sqrt{n}) + n$

   $$\frac{K(n) - K(n - \sqrt{n})}{\sqrt{n}} = \frac{n}{\sqrt{n}}$$

   $K'(n) = \Theta(\sqrt{n})$
   $K(n) = \Theta\left(n^{\frac{3}{2}}\right)$

5. What is the asymptotic complexity of the C++ function `martha(x)` given below, in terms of x? Write a recurrence and solve.

```
float martha(float x)
 {
  assert(x > 0);
  if(x < 1) return 0;
  else return 2*martha(x/2) + x;
 }
```

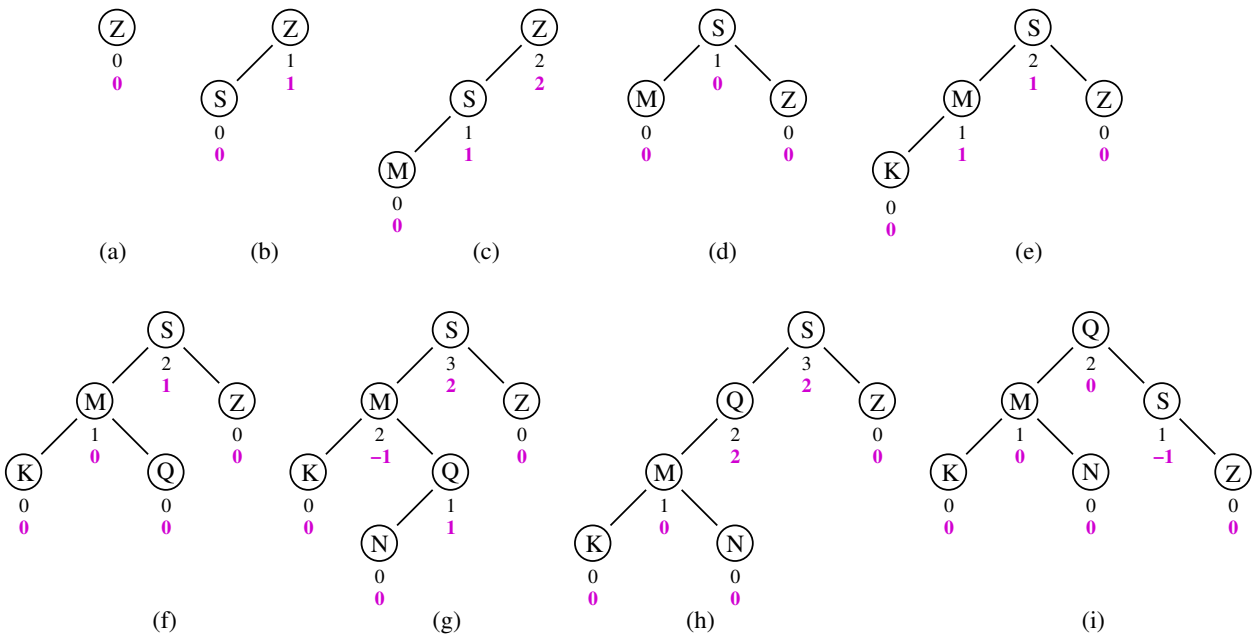
   The recurrence is $M(x) = 2M(x/2) + x$. Thus $M(x) = \Theta(x \log x)$.

6. What is the asymptotic complexity of the C++ function `george(x)` given below, in terms of x? Write a recurrence and solve. Assume that each operation takes constant time.

```
float george(float x)
 {
  assert(x > 0);
  if(x < 1) return 0;
  else return george(3*x/5) + george(4*x/5) + x*x;
 }
```

The recurrence is $G(x) = G(3x/5) + G(4x/5) + x^2$. Thus $G(x) = \Theta(x^2 \log x)$.

7. Starting with an empty AVL tree, insert the letters Z,S,M,K,Q,N in that order. Show the tree after each insertion and each rotation.



In each figure, the black digit shows the height of the node, while the red digit shows the balancd factor. The steps are listed below.

(a) Insert Z, obtaining (a). The tree is balanced.

(b) Insert S, obtaining (b). The tree is balanced.

(c) Insert M, obtaining (c). The tree is unbalanced at Z.

(d) Right rotation at Z, obtaining (d). The tree is now balanced.

(e) Insert K, obtaining (e). The tree is balanced.

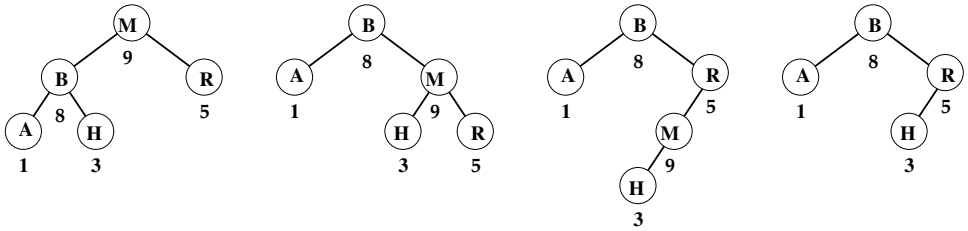
(f) Insert Q, obtaining (f). The tree is balanced.

(g) Insert N, obtaining (g). The tree is now unbalanced at S.

(h) Double right rotation at S. First make a left rotation at M, obtaining (h).

(i) Next, right rotaion at S, obtaining (i). The tree is now balanced.

8. Show the steps of deletion of M from the treap given below.



1. Right rotate to bring B, the highest priority child of M, up.
2. Recursively execute deletion of M. Right rotate to bring R up.
3. Recursively execute deletion of M. Since it has an empty right child, replace the M by its left child H.

9. The following is C++ code for a function that you are familiar with. I have verified that the code compiles and runs and is correct. The value of `guess(x,b)` computed by the program is rounded to six significant figures. That value could be positive, negative or zero. The parameter M is not mathematically related to the function, but is only a programming device to contain the recursion, since otherwise the recursive depth could theoretically be infinite.

```
const int N = 30; // limits the depth of the recursion

float guess(float x, float b, int M)
 {
  assert(x > 0.0 and b > 1.0);
  if(x == 1) return 0;
  else if(M <= 0) return 0; // Really?  Shouldn't it be 1?  Does it matter?
  else if(x < 1) return -guess(1/x,b,M);
  else if(x > b) return 1 + guess(x/b,b,M);
  else if(x < b) return guess(x*x,b,M-1)/2;
  else return 1.0;
 }

float guess(float x, float b)
 {
  cout << "Computing guess(" << x << "," << b << ")" << endl;
  return guess(x,b,N);
 }

int main()
 {
  float x;
  cout << "Enter a positive number x:  ";
  cin >> x;
  cout << endl;
  float b;
```

```
  cout << "Enter a number larger than 1:  ";
  cin >> b;
  cout << endl;
  cout << guess(x,b) << endl;
  return 1;
}
```

What does the function guess(x,b) compute? The code is available on the Assignments page, and can be downloaded into a text file and run. Suggested cases to try: `guess(10,3)` and `guess(0.03125,4)`

For any $x > 0$ and $b > 1$, $\text{guess}(x, b) = \log_b x$, the base b logarithm of x.

10. Fill in the blanks.

    (a) True or false: Open hashing uses open addressing. **False**.

    (b) When two data have the same hash value, that is called a **collision**.

    (c) A **perfect** hash function gives a 1-1 correspondence between the data and the indices of the hash table.

    (d) In closed hashing, if a collision occurs, one of the data uses a **probe** sequence to search for an unused index.

    (e) In open hashing, the data which share a hash value must be stored in a **search structure**. (Choose one of these answers: **search structure**, **priority queue**, **virtual array**, **directed graph**.)

    (f) An optimal binary prefix code for a given weighted alphabet can be computed using **Huffman's** algorithm.

    (g) In an unweighted directed graph, the shortest path between two given vertices can be found by **depth**-first search. (Choose one of these answers: **depth**, **breadth**.)

    (h) Binary search tree sort (or simply *tree sort*) is a fast implementation of **insertion** sort. (Choose of these answers: **selection**, **bubble**, **insertion**, **quick**.)

    (i) A **topological** order of a directed graph G is an ordering of the vertices of G such that vertex x must be come earlier than vertex y in the ordering if there is an arc from x to y,

    (j) The subproblems of a dynamic program must be worked in **topological** order.