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

Answers to Assignment 6: Due Saturday April 5 2025

1. 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) A connected acyclic graph (not digraph) with 25 vertices must have **24** edges.

   (f) 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**.)

   (g) In **cuckoo** hashing, each datum has more than one possible hash value.

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

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

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

   (k) 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,

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

2. 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++)
   ```
   $\Theta(n \log n)$

   (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++)
   ```
   $\Theta(\sqrt{n})$

3. 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) $T(n) = 7T(n/7) + n$

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

   (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: $m = \log n$.

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

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

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

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

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

   $\gamma = 1$ since $2\left(\frac{1}{4}\right) + \frac{1}{2} = 1$
   $F(n) = \Theta(n)$

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

   $\gamma = 1$ since $2\left(\frac{1}{4}\right) + \frac{1}{2} = 1$
   $F(n) = \Theta(n \log n)$

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

$\gamma = 1$ since $2\left(\frac{1}{4}\right) + \frac{1}{2} = 1$
$F(n) = \Theta(n^2)$

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

$\gamma = 2$, since $\left(\frac{3}{5}\right)^2 + \left(\frac{4}{5}\right)^2 = 1$
$F(n) = \Theta(n^2 \log n)$ vskip 0.1in

(e) $F(n) = F(n/3) + 5F(2n/3) + 1$

$\gamma = 4$, since $\left(\frac{1}{3}\right)^4 + 5\left(\frac{2}{3}\right)^4 = 1$

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

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

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

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

   $K'(n) = \Theta(\sqrt{n})$
   $K(n) = \Theta(n^{3/2})$

6. What is the asymptotic complexity of the function `martha(n)` given below, in terms of n? Write a recurrence and solve. (I mean the actual value of `martha(n)`, not the time to compute it.)

```
int martha(int n)
 {
  assert(n >= 0);
  if(n < 1) return 0;
  else return 2*martha(n/2) + n;
 }
```

   $\text{martha}(n) = 2\text{martha}(n/2) + n$
   $\text{martha}(n) = \Theta(n \log n)$

7. What is the asymptotic time complexity of the above code which computes `martha(n)`? Assume that each addition or multiplication takes constant time.

   Let $T(n)$ be the time for the above code to compute martha(n). Then

   $T(n) = 2T(n/2) + 1$ since it only takes 1 step to fetch n.

   $T(n) = \Theta(n)$.

8. If you actually need the value of **martha(n)** and not other values of **martha**, the above recursive code is rather inefficient. Describe a faster method. What is its asymptotic time complexity? Assume that each addition or multiplication takes constant time.

   We can use dynamic programming, which takes $\Theta(n)$ time:

```
martha[0] = 0;
for(int i = 1; i <= n; i++)
 martha[i] = 2*martha[i/2] + i

write martha[n]
```

   We can use memoization, which takes $\Theta(\log n)$ time: Assume memos of the form (i,martha(i)) are stored in a search structure.

```
int martha(int i)
  if (there is a memo (i,m)) return m;
  else
    if(i == 0) m = 0;
    else
      m = martha(i/2) + i;
      store the memo (i,m)
      return m;

write martha(n)
```

9. What is the asymptotic complexity of the function `george(n)` given below, in terms of n? Write a recurrence and solve. (I mean the actual value of `george(n)`, not the time to compute it.)

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

   The recurrence is george(n) = george($3n/5$)+ george($4n/5$) + n^2

   Using the Akra-Brazzi method, $\gamma = 2$.

   The solution is george(n) = $\Theta(n^2 \log n)$.

10. What is the asymptotic time complexity of the above code which computes `george(n)`? Assume that each operation takes constant time.

The recurrence is:

The recurrence is $T(n) = T(3n/5) + T(4n/5) + 1$

Using the Akra-Brazzi method, $\gamma = 2$.

The solution is $T(n) = n^2$.

11. The following table gives two possible hash values for each of a set of 8 data. Can you construct a closed hash table of size 8 which contains all the data?

If so, construct the table. Otherwise, convince me that it's impossible.

Abe	1	4
Bob	7	3
Cec	5	6
Dan	5	7
Eve	1	6
Fay	2	3
Hal	4	8
Ida	8	2

1	~~Abe~~ ~~Eve~~ Abe
2	Fay
3	Bob
4	~~Abe~~ Hal
5	~~Cec~~ ~~Dan~~ Cec
6	~~Cec~~ Eve
7	~~Bob~~ Dan
8	Ida