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

Answers to Examination March 6, 2024

1. Fill in the blanks.

   (a) Any comparison-based sorting algorithm on a file of size n must execute $\Omega(n \log n)$ comparisons in the worst case. Use Ω.

   (b) The asymptotic time complexity of mergesort on an array of length n $\Theta(n \log n)$. (Use Θ.)

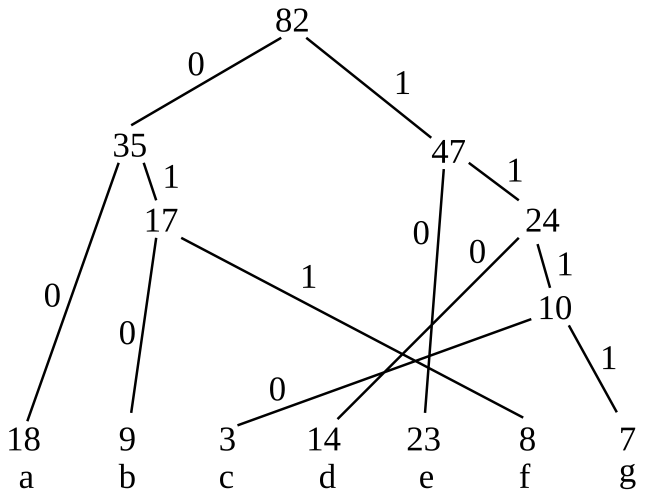
   (c) The (worst case) asymptotic time complexity of treesort on n items is $O(n^2)$.

   (d) If you use a treap, the worst case asymptotic time complexity of treesort on n items is $O(n^2)$.

   (e) If you use an AVL tree, the worst case asymptotic time complexity of treesort on n items is $\Theta(n \log n)$.

2. Find an optimal prefix-free binary code for the following weighted alphabet.

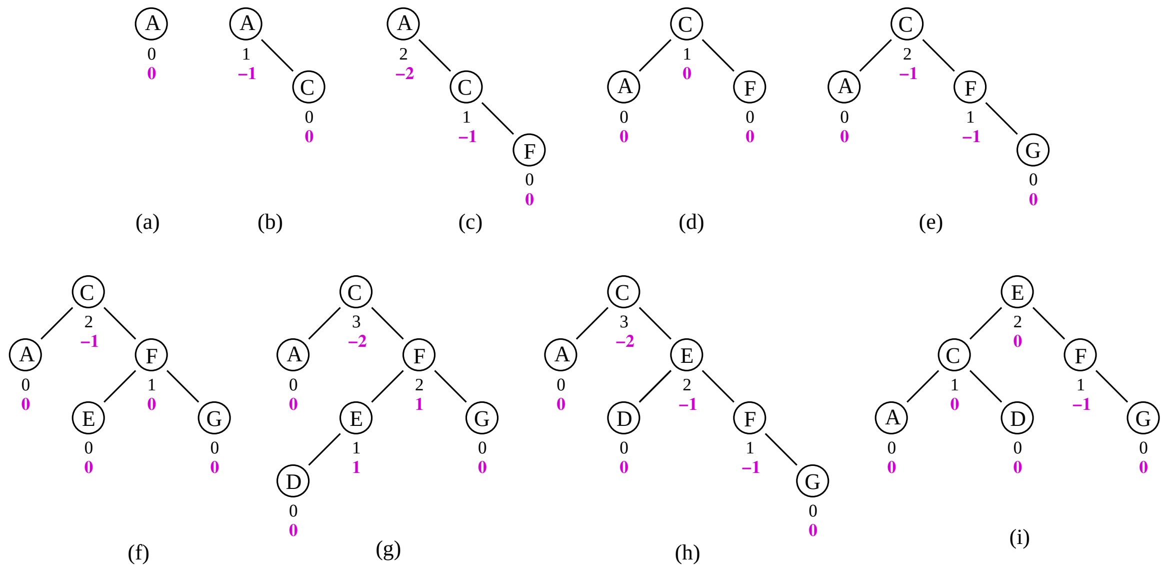
a	18	00
b	9	010
c	3	1110
d	14	110
e	23	10
f	8	011
g	7	1111



3. Given a binary search tree T which has n nodes and height h, what is the time required to find an item is T? Fill in one circle.

   ○ $O(n)$
   [✓ $O(h)$]
   ○ $O(\log n)$
   ○ $O(\log h)$

4. Walk through insertion of the items A, C, F, G, E, D, in that order into an AVL tree.



5. Solve these recurrences.

   (a) $F(n) = 2F(n/4) + \sqrt{n}$
   $F(n) = \Theta(\sqrt{n} \log n)$

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

   (c) $F(n) = F(n/9) + F(4n/9) + \sqrt{n}$
   $F(n) = \Theta(\sqrt{n} \log n)$

6. Write C++ code for the standard $O(n^2)$-time versions of selection sort and insertion sort, on an integer array A of size n. There is more than one correct answer to each of these problems.

```
void swap(int&x, int&y)
 {
  int temp = x;
  x = y;
  y = temp;
 }

void selectionsort()
 { // 4 lines deleted
  for(int i = 0; i < n-1; i++)
   for(int j = i+1; j < n; j++)
    if(A[j] < A[i])
     swap(A[i],A[j]);
 }

void insertionsort()
 { // 4 lines deleted
  for(int i = 0; i < n-1; i++)
   for(int j = i+1; j > 0 and A[j] < A[j-1]; j--)
    swap(A[j],a[j-1]);
 }
```

7.
```
void quicksort(int first, int last)
  // sorts the subarray A[first .. last]
 {
  if(first < last) // otherwise there is at most one entry
   {
    int mid = (first+last)/2;
    swap(A[first],A[mid]);
    int pivot = A[first];
    int lo = first;
    int hi = last;
    // loop invariant holds here
    while(lo < hi) // the partition loop
     {
      // loop invariant holds here
      if(A[lo+1] <= pivot) lo++;
      if(lo < hi and A[hi] >= pivot) hi--
      if(lo < hi) and A[lo+1] > pivot and A[hi] < pivot)
       swap(A[lo+1],A[hi]);
     }
     // loop invariant holds
    swap(A[first],A[lo]);
    // now A[lo] = pivot
    quicksort(first,lo-1);
    quicksort(lo+1,last);
   }
 }

int main()
 {
  quicksort(0,N-1);
  return 1;
 }
```

What is the loop invariant of the partition loop?

lo $\leq$ hi and A[i] $\leq$ pivot for all first $\leq$ i $\leq$ lo and A[j] $\geq$ pivot for all hi $<$ i $\leq$ last.

8. Write an $O(n)$-time algorithm which, given a sequence $\sigma = (x_1, x_2, \dots x_n)$ of positive numbers, computes the maximum sum of any subsequence of σ which contains no two consecutive terms of σ. For example, if $\sigma = (1, 4, 2, 1, 5, 3, 6, 7, 4)$, the maximum sum of such a subsequence is $4 + 5 + 6 + 4 = 19$.

   Define S[i] to be the maximum sum of any subsequence of $x_1, \dots x_i$. The following program computes all S[i] and returns S[n].

```
S[0] = 0;
S[1] = $x_1$;
for(int i = 2; i $\leq$ n; i++)
   S[i] = max(S[i−1], S[i−2]+$x_i$);
return S[n];
```

   For the example sequence given above, the values of S[i] are:

$S[0] = 0$	$S[5] = 9$
$S[1] = 1$	$S[6] = 9$
$S[2] = 4$	$S[7] = 15$
$S[3] = 4$	$S[8] = 16$
$S[4] = 5$	$S[9] = 19$

9. In class, I presented three methods for solving the false overflow problem for the array implementation of queue, where items are inserted at one end and deleted from the other. There is a fourth method which is not available in every modern programming language; for example, it is not available in Pascal. What are those methods? (Do not give details. Just name each method in a word or a short phrase.)

   1. Slide
   2. Wrap
   3. Make the array big enough so that false overflow won't happen.

   The fourth method, which is not available in all languages, is to expand the array.

10. Find the asymptotic time complexity of each of these code fragments in terms of n, using Θ notation.

   (a)
```
for(int i = 0; i*i < n; i++)
```

   $\Theta(\sqrt{n})$

   (b)
```
for(int i = 1; i < n; i = 2*i)
 for(int j = 2; j < i; j = j*j);
```

   Substituting: $k = \log i$, $m = \log n$, $\ell = \log j$, we get

```
for(int k = 0; k < m; k++)
 for(int l = 1; l < k; l = 2*l)
```

   The answer is $\Theta(m \log m) = \Theta(\log n \log\log n)$

11. Let $W_1 = 1$, $W_2 = 2$, and $W_n = 2W_{n-1} + 3W_{n-2}$ for $n \geq 2$. For example, $W_3 = 7$ and $W_4 = 20$. Find a constant K such that $W_n = \Theta(K^n)$.

    Assume $W_n = K^n$. Then $K^n = 2K^{n-1} + 3K^{n-2}$. Dividing both sides by K^{n-2}, we obtain the quadratic equation $K^2 = 2K + 3$, and the only positive solution is $K = 3$.

12. The following function computes $x * n$. Find a loop invariant of the while loop.

```
float prod(float x, int n) // input condition: n >= 0
 {
  int m = n;
  float y = x;
  float z = 0.0;
  while(m > 0)
   {
    if(m%2) z = z+y;
    m = m/2;
    y = y+y;
   }
  return z;
 }
```

    $x * n = z + y * m$

13. The following function computes x^n. Find a loop invariant of the while loop.

```
float pwr(float x, int n) // input condition: x > 0 and n >= 0
 {
  int m = n;
  float y = x;
  float z = 1.0;
  while(m > 0)
   {
    if(m%2) z = z*y;
    m = m/2;
    y = y*y;
   }
  return z;
 }
```

    $x^n = z * y^m$

14. Fill in the blanks.

    (a) **Binary search** is a divide-and-conquer search algorithm which only works on a sorted list.

    (b) **Linear search** is an $O(n)$-time search algorithm, generally used only when n is small.

15. Write the prefix expression equivalent to the infix epression $-a * b - (-c - d) \wedge e$
(Don't forget that $\wedge$ means exponentiation.)

$- * \sim ab \wedge - \sim cde$

16. Walk through the stack algorithm to change the infix expression $-a + b \wedge c \wedge -f$ to postfix. Show the stack at each step.

stack	input	output
	$-a+b\wedge c\wedge -f$	
$\sim$	$a+b\wedge c\wedge -f$	
$\sim$	$+b\wedge c\wedge -f$	a
	$+b\wedge c\wedge -f$	$a\sim$
$+$	$b\wedge c\wedge -f$	$a\sim$
$+$	$\wedge c\wedge -f$	$a\sim b$
$+\wedge$	$c\wedge -f$	$a\sim b$
$+\wedge$	$\wedge - f$	$a\sim bc$
$+\wedge$	$\wedge - f$	$a\sim bc$
$+\wedge\wedge$	$-f$	$a\sim bc$
$+\wedge\wedge\sim$	f	$a\sim bc$
$+\wedge\wedge\sim$		$a\sim bcf$
$+\wedge\wedge$		$a\sim bcf\sim$
$+\wedge$		$a\sim bcf\sim\wedge$
$+$		$a\sim bcf\sim\wedge\wedge$
		$a\sim bcf\sim\wedge\wedge+$

17. In this problem, assume that it takes one time step to compute any addition or multiplication.

Consider the following recursive C++ function.

```
int f(int n)
 {
  if(n <= 0) return 0;
  else
   return f(n/6) + f(n/3) + f(n/2) + n;
 }
```

(a) Write a dynamic program which computes f(0) ... f(n) by dynamic progrmming, storing them in the following array.

`int f[n+1];`

```
f[0] = 0
for(int i = 1; i ≤ n; i ++);
    f[i] = f[n/6] + f[n/3] + f[n/2] + n;
```

What is the time complexity of your program?

$\Theta(n)$

(b) Write a recurrence for $f(n)$ and solve it, giving an asymptotic answer.

```
f(n) = f(n/6) + f(n/3) +f(n/2) + n
```

By the generalized master theorem (Akra-Brazzi) the solution is
f(n) $= \Theta(n \log n)$

(c) Let t(n) be the time it takes for the above code to compute f(n). Write a recurrence for t(n) and solve it, giving an asymptotic answer.

```
t(n) = t(n/6) + t(n/3) +t(n/2) + 1
```

By the generalized master theorem (Akra-Brazzi) the solution is
t(n) $= \Theta(n)$

(d) If you only need the value of f(n), instead of f(i) for all i in $0 \ldots$ n, you could use memoization. How many memos would you need to compute and store? Give an asymptotic answer, in terms of n. Hint: it's less than n.

You need to compute and store $f(n/(2^i 3^j)$ for all values of i, j where $2^i 3^j \leq n$. There are approximately $(\log_2 n)(\log_3 n)/2 = \Theta(\log^2 n)$ such pairs.