CS 477/677 Answers to Study Guide for Examination October 2024

Sun Oct 20 08:28:22 PM PDT 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.

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

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

   (d) The asymptotic time complexity of the Floyd Warshall algorithm on a weighted directed graph with n vertices and m arcs is $\Theta(n^3)$.

   (e) The asymptotic time complexity of Dijkstra's algorithm on a weighted directed graph with n vertices and m arcs is $O(m \log n)$

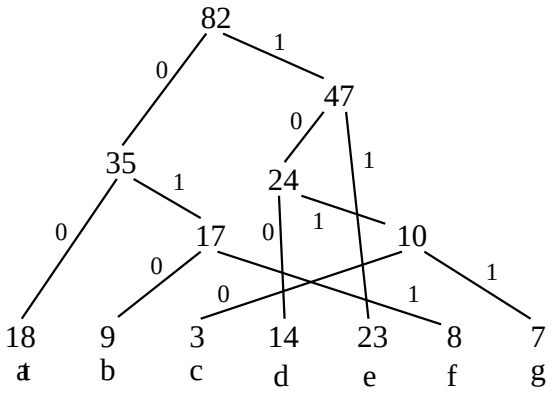
   (f) The asymptotic time complexity of Treesort of n items is $O(n^2)$ in the worst case, but if you use a balancing scheme for the tree, you can expect that the asymptotic time complexity is $\Theta(n \log n)$.

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

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

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

a	18	00
b	9	010
c	3	0010
d	14	100
e	23	11
f	8	011
g	7	1011



3. Given a binary search tree T which has n nodes and height h, what is the worst case time complexity of search? Only one of these answers is correct.

   ◯ $O(n)$
   ⊗ $O(h)$
   ◯ $O(\log n)$
   ◯ $O(\log h)$

4. Solve these recurrences

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

   $F(n) = \Theta(\sqrt{n} \log n)$

(b) $F(n) = F(n/4) + \sqrt{n}$

$F(n) = \Theta(\sqrt{n})$

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

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

(d) $F(n) = F(n - \sqrt{n}) + n^2$

$F(n) = \Theta(n^{5/2}$

(e) $F(n) = F(\log n) + 1$

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

5. For each of the following recurrences, substitute $m = \log n$.

(e) $F(n) = F(\sqrt{n}) + 1$

Substitute $m = \log n$, $G(m) = G(\log n) = F(n)$
$F(\sqrt{n}) = G(\log \sqrt{n}) = G(\log n/2) = G(m/2)$
$G(m) = G(m/2) + 1$
$F(n) = G(m) = \Theta(\log m) = \Theta(\log \log n)$

(f) $F(n) = 4F(\sqrt{n}) + 1$

Substitute $m = \log n$, $G(m) = G(\log n) = F(n)$
$F(\sqrt{n}) = G(\log \sqrt{n}) = G(\log n/2) = G(m/2)$
$G(m) = 4G(m/2) + 1$
$F(n) = G(m) = \Theta(m^2) = \Theta(\log^2 n)$

(g) $F(n) = 4F(\sqrt{n}) + \log^2 n$

Substitute $m = \log n$, $G(m) = G(\log n) = F(n)$
$G(m) = 4G(m/2) + m^2$
$F(n) = G(m) = \Theta(m^2 \log m) = \Theta(\log^2 n \log \log n$

6. 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++) Replace $i^2 < n$ by $i < \sqrt{n}$ and we have
for(int i = 0; i < $\sqrt{n}$; i++)

The complexity is thus $\Theta(\sqrt{n})$

(b)
```
for(int i = 0; i < n; i++)
  for(int j = 1; j < i; j = 2*j);
```

Substitute $k = \log j$

```
for(int i = 1; i < n; i++)
  for(int k = 0; k < log i; k++);
```

Approximate by an integral:
$\int_{x=1}^{n} \int_{y=0}^{lnx} dydx = \int_{x=1}^{n} \ln x dx = \Theta(n \log n)$

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

Substitute $k = \log j$ and we have

```
for(int i = 1; i < n; i++)
 for(int k = log i; k < log n; k++);
```

Approximate by an integral:
$\int_{x=1}^{n} \int_{y=\ln x}^{\ln n} dydx = \int_{x=1}^{n} (\ln n - \ln x)dx = |x \ln n - x \ln x + x|_{x=1}^{n} = \Theta(n)$

(d) `for(float x = n; x > 2.0; x = sqrt(x))` (`sqrt(x)` returns the square root of `x`.)

Let $y = \log x$. Substituting, we have: `for(float y = log n; y > 1.0; y = y/2)`

Let $z = \log y$. Substituting, we have: `for(float z = log log n; z > 0.0; z = z-1.0)`
The time complexity is $\Theta(\log \log n)$

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

Substitute $k = \log i$, $l = \log j$, $m = \log n$. We have:

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

We now simply copy the answer from problem 6(b) above, and we obtain the answer $\Theta(m \log m) = \Theta(\log n \log \log n)$.

7. 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.

```
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 = 1; i < n; i++)
   for(int j = i; j > 0; j--)
    if(A[j] < A[j-1])
     swap(A[j],A[j-1]);
 }
```

8. 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.)

   (a) Slide. When the rear of the queue overflows the array, slide everything toward the front.

   (b) Wrap. Make the queue circular.

   (c) Bigger. If you happen to know that the total number of insertions during the program will not exceed N, simply make the array size N.

   (d) Increase. Redefine the array to be larger. Not available in all programming languages.

9. Write pseudocode for an $O(n)$-time algorithm which, given a sequence $\sigma = (x_1, x_2, \ldots x_n)$ 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 $4+5+6+4 = 19$.

   We say a subsequence is *legal* if it does not include any two consecutive terms. We let a_i be the maximum sum of any legal subsequence which ends at x_i. The following pseudocode finds the maximum sum of any legal subsequence.

   $a_1 = x_1$
   $a_2 = x_2$
   $a_3 = x_1 + x_3$
   for all i from 4 to n
     $a_i = x_i + \max(a_{i-2}, a_{i-3})$
   The maximum sum of any legal subsequence is $max(a_n, a_{n-1})$.

10. 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 $W_n = K^n$ for all n. (This is false, but it's close enough to give you the right value of K.) Then $K^n = 2K^{n-1} + 3K^{n-2}$. Dividing by K^{n-2}, we get $K^2 = 2K + 3$, which is a quadratic equation, and the solution is $K = 3$ or $K = -1$. Since the sequence $\{W_n\}$ is increasing, $K = 3$.

11. 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;
 }
```

The loop invariant is $x * n = y * m + z$. After the loop exits, $m = 0$, hence $x * n = z$.

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

$-* \sim ab \wedge - \sim cd$

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;
 }
```

The loop invariant of the while loop is $x^n = y^m * z$

14. Write pseudocode for the Floyd Warshall algorithm on a weighted directed graph. Assume the vertices are the integers from 0 to $n - 1$ and W$[i, j]$ is the weight of the arc from i to j. If there is no such arc,

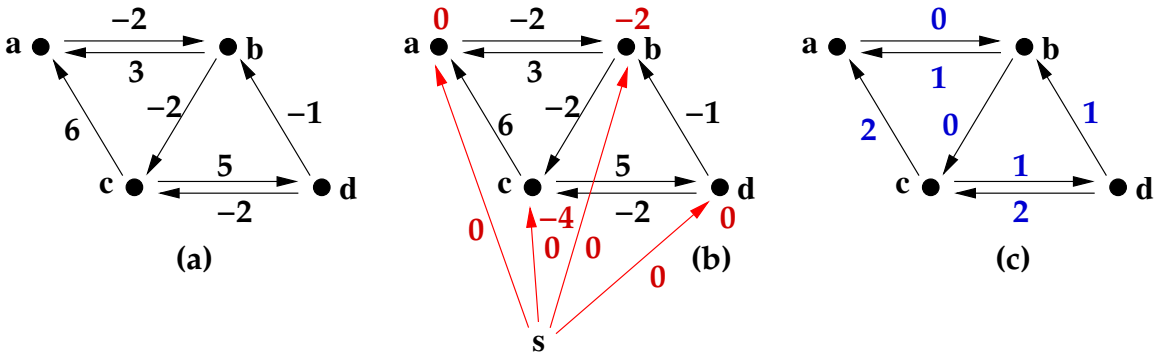
W$[i, j] = \infty$. Your code should calculate all V$[i, j]$, the smallest weight of any path from i to j, and back$[i, j]$, the back pointer for that path.

```
For all i and all j
  V[i,j] = W[i,j] and back[i,j] = i.
For all i V[i,i] = 0
  For all j
   For all i
    For all k
     {
      temp = V[i,j] + V[j,k]
      if(temp < V[i,k])
       {
        V[i,k] = temp
        back[k] =back[j,k]
       }
     }
```

15. Write pseudocode for the Bellman Ford algorithm on a weighted directed graph. I won't specify the notation as I did for Floyd Warshall; I will leave that task to you. Be sure to incorporate the shortcut.

```
V[0] = 0
bool finished = false
while(not finished)
  {
   finished = true;
   For all i,j such that W[i,j] < ∞
    {
     temp = W[i,j] + V[i]
     if(temp< V[j])
      {
       V[j] = temp
       finished = false;
       back[j] = i
      }
    }
  }
```

16. We need to execute Johnson's algorithm on the weighted digraph shown in (a). Since there are negative weight edges, the weights of the edges are first adjusted, so that all will be non-negative. Show the adjusted weights in (c). You can use (a) and (b) for your work. Do not complete Johnson's algorithm.



17. Here is C++ code for quicksort on a given array $A[N]$ of type int, as given in the handout sorting.pdf. I have tested the code.

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

void quicksort(int first, int last)    // input condition: first <= last
  // sorts the subarray A[first .. last]
 {
  if(first < last) // otherwise there is only 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
    while(lo < hi) // the partition loop
     {
       // loop invariant holds
      while(A[lo+1] < pivot)lo++;
      while(A[hi] > pivot)hi--;
      if(lo+1 < hi)
       {
        swap(A[lo+1],A[hi]);
        lo++;
        hi--;
       }
      else if(lo+1 == hi) hi--;
     }
     // loop invariant holds
    swap(A[first],A[lo]);
    // now A[lo] = pivot
```

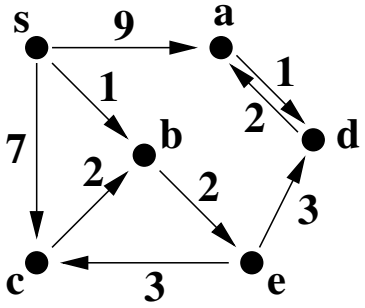
```
    if(first < lo) quicksort(first,lo-1);
    if(lo+1 < last) quicksort(lo+1,last);
   }
 }

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

What is the loop invariant of the partition loop?

For all first <= i <= hi, $A[i]$ ≤pivot, and for all hi < i <= last, $A[i]$ ≥pivot.

18. Use Dijkstra's algorithm to solve the single source minpath problem for the weighted digraph shown below.



s	a	b	c	d	e
0	~~9~~ 8	1	~~7~~ 6	6	3
*	~~s~~ d	s	~~s~~ e	e	b

19. Find the maxflow and mincut in a weighted directed graph. Figure (a) shows the weighted digraph G. The goal is to maximize the flow from S to T. Figure (b) shows the initial flow. Draw the residual graph in Figure (c). Use the remaining space for your work. Be sure to indicate the mincut.

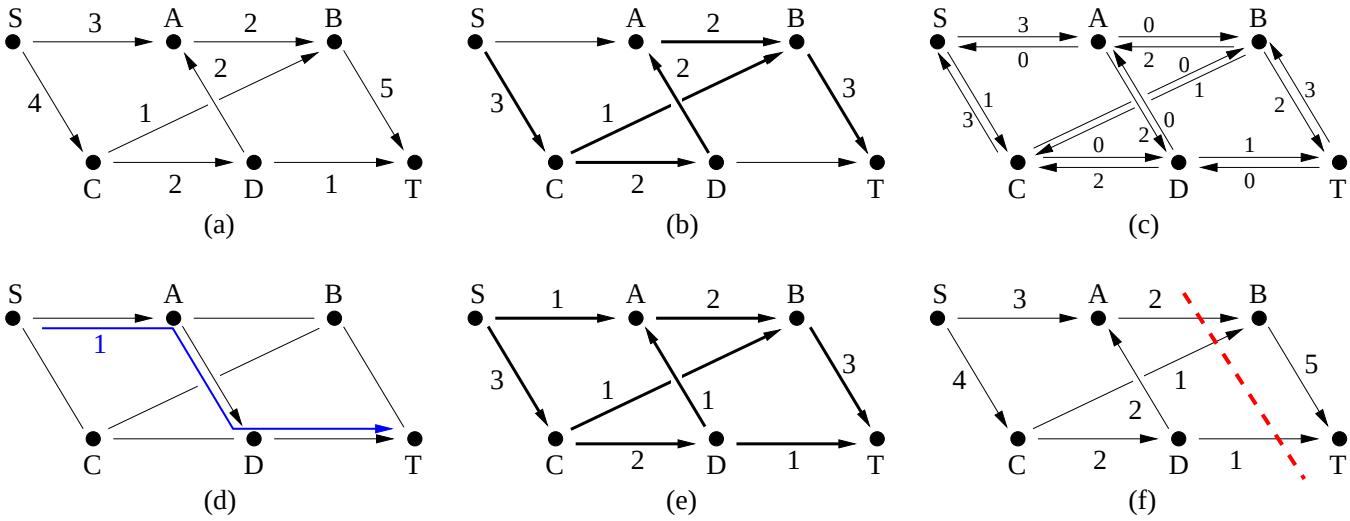


Figure (c) shows the augmenting flow of 1 unit from S to T. Figure (d) shows the current flow of 4 after adding the augmentation. Figure (e) shows a cut of 4 on the original graph, proving that 4 is optimal by the maxflow/mincut theorem.