University of Nevada, Las Vegas Computer Science 477/677 Fall 2025

Answers to Assignment 3: Due Saturday October 4, 2025

Follow our TA Louis DuMontet's (`dumontet@unlv.nevada.edu`) instructions on how to turn in the assignment.

Name:_____

You are permitted to work in groups, get help from others, read books, and use the internet.

1. Fill in the blanks.

   (a) No good programmer would ever use an unsorted list as a search structure. **F** (True or False.)

   (b) Heapsort is a fast version of **selection sort**

   (c) Treesort is a fast version of **insertion sort**

   (d) Shell sort is inspired by **bubblesort** but is faster.

   (e) The items of a priority queue represent **unfulfilled obligations**

2. Walk throught the computation of polyphase mergesort with the initial array QZRLFKNSHTWDMVJE.

   Q Z R L F K N S H T W D M V J E

   Q Z L H T W J
   R F K̇ N S D M̈ V E

   O R Z D H M T V W
   F K L N S E J

   F K L N O R S Z
   D E H J M T V W

   D E F J K L H M N O R S T V W Z

3. Solve the following recurrences. Express the answers using Θ notation. Problems (a)–(f) use the Bently-Biostein-Saxe method, the master theorem. Problems (g)–(j) use the anti-derivative method. Problems (k)-(m) use the Akra-Bazzi method. Problem (n) uses the master theorem, after the substitution $n = \log m$.

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

   $A = 2$, $B = 2$, $C = 1$. $\log_B A = 1 = C$, thus
   $F(n) = \Theta(n \log n)$

   (b) $F(n) = 2F(n/2) + 1$

   $A = 2$, $B = 2$, $C = 0$. $\log_B A = 1 > C$, thus
   $F(n) = \Theta(n)$

(c) $F(n) = F(n/2) + 1$

$A = 1$, $B = 2$, $C = 0$. $\log_B A = 0 = C$, thus
$F(n) = \Theta(\log n)$

(d) $F(n) = 9F(n/3) + n$

$A = 9$, $B = 3$, $C = 1$. $\log_B A = 2 > C$, thus
$F(n) = \Theta(n^2)$

(e) $F(n) = 3F(n/3) + n$

$A = 3$, $B = 3$, $C = 1$. $\log_B A = 1 = C$, thus
$F(n) = \Theta(n \log n)$

(f) $F(n) = F(n/3) + n$

$A = 1$, $B = 3$, $C = 1$. $\log_B A = 0 < C$, thus
$F(n) = \Theta(n)$

(g) $F(n) = F(n-1) + n$

$$\frac{F(n) - F(n-1)}{1} = n$$
$$F'(n) = n$$
$$F(n) = \Theta(n^2)$$

(h) $F(n) = F(n - \sqrt{n}) + n$

$$\frac{F(n) - F(n-\sqrt{n})}{\sqrt{n}} = \frac{n}{\sqrt{n}}$$
$$F'(n) = n^{\frac{1}{2}}$$
$$F(n) = n^{\frac{3}{2}}$$

(i) $F(n) = F(n - \log n) + \log^2 n$

$$\frac{F(n) - F(n-\log n)}{\log n} = \frac{\log^2 n}{\log n}$$
$$F'(n) = \log n$$
$$F(n) = \Theta(n \log n)$$

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

$$\frac{F(n) - F(n-\sqrt{n})}{\sqrt{n}} = \frac{1}{\sqrt{n}}$$
$$F'(n) = n^{-\frac{1}{2}}$$
$$F(n) = n^{\frac{1}{2}}$$

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

$\alpha_1 = 1$, $\beta_1 = \frac{3}{5}$, $\alpha_2 = 1$, $\beta_2 = \frac{4}{5}$, $\gamma = 2$.
$\alpha_1 \beta_1^\gamma + \alpha_2 \beta_2^\gamma = \left(\frac{3}{5}\right)^2 + \left(\frac{4}{5}\right)^2 = 1$, and thus $F(n) = \Theta(n^\gamma \log n) = \Theta(n^2 \log n$

(l) $F(n) = F(n/2) + F(n/3) + n$

$\alpha_1 = 1,\ \beta_1 = \frac{1}{2},\ \alpha_2 = 1,\ \beta_2 = \frac{1}{3},\ \gamma = 1.$
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = \left(\frac{1}{2}\right) + \left(\frac{1}{3}\right) = \frac{5}{6} < 1$, and thus $F(n) = \Theta(n^\gamma) = \Theta(n)$

(m) $F(n) = F(12n/13) + F(5n/13) + n$

$\alpha_1 = 1,\ \beta_1 = \frac{12}{13},\ \alpha_2 = 1,\ \beta_2 = \frac{5}{13},\ \gamma = 1.$ $\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = \left(\frac{12}{13}\right) + \left(\frac{5}{13}\right) = \frac{17}{13} > 1$. Thus, we must find δ such that $\left(\frac{12}{13}\right)^\delta + \left(\frac{5}{13}\right)^\delta = 1$. We find that $\delta = 2$, and thus $F(n) = \Theta(n^\delta) = \Theta(n^2)$.

(n) $F(n) = 2F(n-1) + 1$ **Think.**

Substitute $n = \log m$. Thus $\log \frac{m}{2} = \log m - \log 2 = n - 1$.
Let $G(m) = F(n)$. Our new recurrence is $G(m) = 2G(m/2) + 1$, and thus, by the master theorem, $F(n) = G(m) = \Theta(m) = \Theta(2^n)$.

5. The following is a partial array implementation of the ADT stack of integer. Fill in the missing code for empty, push, and pop.

```
struct stak
 {
  int item[N];
  int size;
 };

void init(stak&S)
 {
  S.size = 0;
 }

bool empty(stak S)
 {
  return S.size == 0;
 }

void push(stak&S, int newitem)
 {
  assert(S.size < N);
  S.item[S.size] = newitem;
  S.size++;
 }

int pop(stak&S)
 {
  assert(not empty(S));
  S.size--;
  int rslt = S.item[S.size];
 }
```

6. State the asysmptotic time complexity, in terms of n, of each of these code fragments. Use Θ notation.

   (a)
   ```
   for(int i = 2; i < n; i=i*i)
    cout << "Hello world." << endl;;
   ```

   $\Theta(\log \log n)$

   (b)
   ```
   for(int i = 2; i*i < n; i++)
    cout << "Hello world." << endl;;
   ```

   $\Theta(\sqrt{n})$

   (c) In the following problem, sqrt means square root.

   ```
   for(int i = n; i > 2; i = sqrt{i})
    cout << "Hello world." << endl;;
   ```

   $\Theta(\log \log n)$

   (d)
   ```
   for(int i = 1; i < n; i++)
    for(int j = 1; j < n; j = 2*j)
     cout << "Hello world." << endl;;
   ```

   $\Theta(n \log n)$

   (e)
   ```
   for(int i = 1; i < n; i = 2*i)
    for(int j = 0; j < i; j++)
     cout << "Hello world." << endl;;
   ```

   $\Theta(n)$

7. Write pseudocode to solve the following dynamic programming problem. Given a row of coins, each of which has a positive value, find the maximum value of a set of those coins, given that the set contains no two coins which are adjacent in the row.

   Let V[i] be the value of the i$^{\text{th}}$ coin.

   A[1] = V[1]
   A[2] = max(A[1],V[2])
   for i from 3 to n:
   A[i] = max(A[i-1],V[i]+A[i-2])
   "The answer is A[n]"

   In the second method, we compute A[i] be the maximum value of any legal set of coins which ends at the i$^{\text{th}}$ coin.

   A[1] = V[1]
   A[2] = V[2]
   A[3] = V[3] + V[1]
   for i from 4 to n:
   A[n] = V[n] + max(A[i-2],A[i-3])
   "The answer is max(A[n-1],A[n])"

8. Write pseudocode to solve the following dynamic programming problem. Given a row of coins, each of which has a positive value, find the maximum value set of those coins, given that the set contains no three coins which are adjacent in the row. For example, the set could contain coins 1,2,4,5. This problem is more complex than Problem 7.

   I will give two methods. In the first method, we compute A[i] to be the maximum value of any legal subset of the first i coins, and let B[i] be the maximum value of any legal subset of the first i coins that does not include the (i-1)st coin.

   B[1] = V[1]
   A[1] = B[1]
   B[2] = V[2]
   A[2] = V[2]+B[1]
   for i from 3 to n
      B[i] = V[i]+A[i-2]
      A[i] = max(A[i-1],B[i],V[i]+B[i-1])
   "The answer is A[n]"

   In the second method, we compute A[i] be the maximum value of any legal set of coins which ends at the ith coin, and B[i] the maximum value of any legal set of coins which ends at the i^{th} coin and does not include the (i-1)st coin.

   B[1] = V[1]
   A[1] = B[1]
   B[2] = V[2]
   A[2] = V[2]+A[1]
   A[2] = V[2]+A[1]
   for i from 3 to n
      B[i] = V[i]+A[i-2]
      A[i] = max(B[i],V[i]+B[i-1])
      "The answer is max(A[n-1],A[n])"

The following C++ code contains an implementation of binary search tree. I have deleted most of the code, leaving only what you need. Fill in the recursive code for the subprograms inorder, preorder and postorder, as well as the functions hite and nmbr, which compute the height and number of nodes of the binary search tree. I deleted code for level order. Below is the output of my program.

```
const int N = 20;

struct treenode;
typedef treenode*tree;
struct treenode
 {
  int item;
  tree left;
  tree right;
 };

tree mainroot;

void inorder(tree t)
 {
  if(t)
   {
    inorder(t->left);
    cout << t->item << " ";
    inorder(t->right);
   }




 }


void preorder(tree t)
 {
  if(t)
   {
    cout << t->item << " ";
    preorder(t->left);
    preorder(t->right);
   }
```

```
 }

void postorder(tree t)
 {
  if(t)
   {
    postorder(t->left);
    postorder(t->right);
    cout << t->item << " ";
   }




 }

int hite(tree t) // height
 // empty tree has hite -1
 {
  if(t)
   return 1+maxint(hite(t->left),hite(t->right));
  else return -1;

 }

int nmbr(tree t) // number of nodes
 {
  if(t)
   return 1+numbr(t->left)+numbr(t->right);
  else return 0;
 }

void insert(tree&t,int newitem)
 {
  if(t == 0)
   {
    t = new treenode();
    t->item = newitem;
   }
  else if(newitem < t->item) insert(t->left,newitem);
  else insert(t->right,newitem);
 }
```

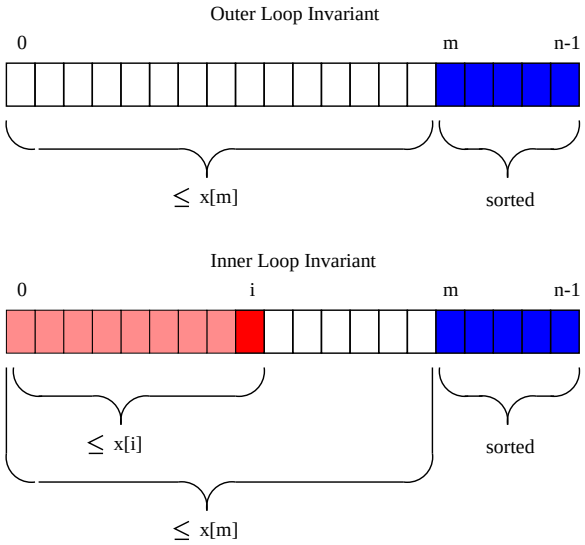
```
  void insertall(tree root); // the code is deleted
  void writelevelorder(tree root); // the code is deleted

  int main()
   {
    insertall(mainroot); cout << endl; // writes items in order of insertion
    cout << "height = " << hite(mainroot();
    cout << "number of nodes = " << nmbr(mainroot();
    writeinorder(mainroot); cout << endl;
    writepreorder(mainroot); cout << endl;
    writepostorder(mainroot); cout << endl;
    writelevelorder(mainroot); cout << endl;
    return 1;
   }

83 26 37 35 33 35 56 22 79 11 42 77 60 89 33 86 70 46 62 56
height = 8
number of nodes  = 20
11 22 26 33 33 35 35 37 42 46 56 56 60 62 70 77 79 83 86 89
83 26 22 11 37 35 33 33 35 56 42 46 79 77 60 56 70 62 89 86
11 22 33 33 35 35 46 42 56 62 70 60 77 79 56 37 26 86 89 83
83 26 89 22 37 86 11 35 56 33 35 42 79 33 46 77 60 56 70 62
```

You must also turn in the code in a form that the TA can execute, such as:

4. If x is an array of length n, let $x[i \ldots j]$ be the set of all $x[k]$ for $i \leq k \leq j$. Thus $x[i \ldots j] = \emptyset$ if $j < i$, while $\min(x[0 \ldots n-1])$ and $\max(x[0 \ldots n-1])$ are the minimum and maximum entries of x, respectively.

The following is C++ code for bubblesort. Write a loop invariant for each of the two loops.

```
void sort(int x[n])
 {
  int m = n;
  while(m > 0)
   {
    int i = 0;
    while(i+1 < m)
     {
      if(x[i+1] < x[i])
       swap(x[i+1],x[i]);
      i = i+1;
     }
    m = m-1;
   }
```

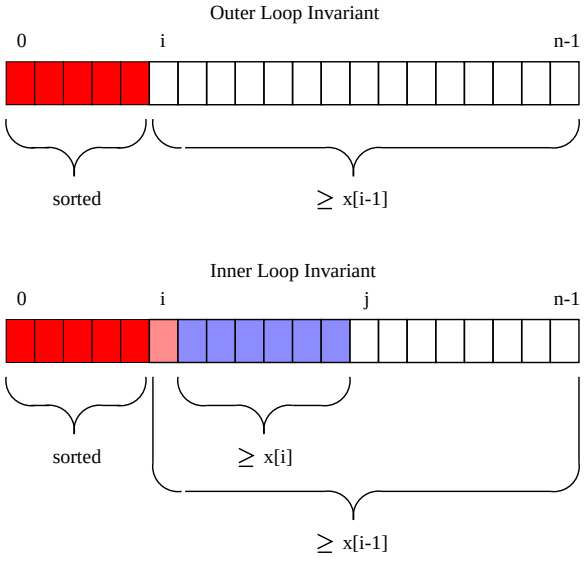


The loop invariant of the outer loop is
$m \geq 0$ and $\max(x[0 \ldots k]) = x[k]$ for all $m \leq k < n$

The loop invariant of the inner loop is
$i < m$ and $\max(x[0 \ldots i]) = x[i]$

The following is C++ code for selection sort. Write the loop invariant for each of the two loops.

```
void sort(int x[n])
 {
  int i = 0;
  while(i < n-1)
   {
    int j = i+1;
    while(j < n)
     {
      if(x[j] < x[i])
       swap(x[i],x[j]);
      j = j+1;
     }
    i = i+1;
   }
 }
```



The loop invariant of the outer loop is
$i \geq 0$ and $\min(x[k \ldots n-1]) = x[k]$ for all $0 \leq k < i$

The loop invariant of the inner loop is
$j > i$ and $\min(x[k \ldots n-1]) = x[k]$ for all $k < i$ and $x[k] \geq x[i]$ for all $i < k < j$