**University of Nevada, Las Vegas Computer Science 477/677 Spring 2022**

**Answers to Assignment 4: Due Monday March 21, 2022, Midnight**

1. True, False, or Open.

   (a) **F** Polyphase mergesort uses dynamic programming.

   (b) **T** Polyphase mergesort uses divide and conquer.

   (c) **T** Dijkstra's algorithm uses dynamic programming.

   (d) **T** The Bellman Ford algorithm uses dynamic programming.

   (e) **T** The Floyd-Warshall algorithm uses dynamic programming.

   (f) **F** A binary search tree is an example of a priority queue.

   (g) **F** A minheap is an example of a search structure.

2. Fill in each blank.

   (a) _____ algorithm requires that no arc have negative weight.

   (b) In a **queue** implemented as a list, items are inserted at one end of the list and deleted from the other end.

   (c) Suppose $G$ is a strongly connected weighted digraph with source vertex 0, which has $n$ edges and $m$ arcs, and where there is an integer $p$ such that for any vertex $i$, the least weight path from 0 to $i$ has at most $p$ edges. What is the worst case time complexity of the Bellman Ford algorithm on $G$, if the code is designed to terminate once the final results are obtained?

   Give an asymptotic answer. $O(mp)$

   (d) Can you name a shortest path algorithm which permits a negative weight loop? **No**

   (e) What is the best-known asymptotic time complexity of any selection algorithm on an unsorted array of size $n$? $O(n)$

3. For each operator, identify its associated abstract data type. The only answers permitted are stack, array, minheap, queue, search structure.

   (a) **stack** push.

   (b) **minheap** deletemin.

   (c) **search structure** find.

   (d) **array** fetch.

   (e) **stack** pop.

   (f) **array** store.

4. Name each of these seven algorithms.

   (a) **quicksort** Pick an element $P$ from a set $S$, then partition S into two parts: those items which are less than $P$ and those greater than $P$. Recorsively sort each part, and combine them to form a sorted list.

   (b) **mergesort** Divide a set $S$ abitrarily into two equal parts. Recursively sort each part, then combine the two sorted parts to obtain a sorting of $S$.

   (c) **binary search** Given a sorted set $S$ and an item $x$, you need to determine whether $x \in S$. Pick one element, say $m$, out of $S$. If $m = x$, you are done. If $m < x$, discard $m$ and all items of $S$ which are greater then $m$, while if $x > m$, discard $m$ all items which are all items of $S$ which are less then $m$. Keep doing this until you either find $x$ or you have discarded all items of $S$.

   (d) **treesort** Given a set $S$, create an empty binary search tree $T$. Insert the items of $S$ into $T$ one at a time. Finally, visit and print the items of $T$ in left-to-right order, also called inorder.

   (e) **selection sort** Given a set $S$, delete the least element of $S$ and print it. Then delete the least remaining element of $S$ and print it. Keep going until you have deleted and printed all elements of $S$.

   (f) **bubblesort** All items of $S$ are in a row. Agents run up and down the row, swapping any two adjacent items if the one on the right is less than the one on the left. Keep going until no more swaps are possible.

   (g) **radix sort** You are grading a large number of exams, each of which is labeled with the ID of a student. Each ID consists of five numerals, and no two students have the same ID. You separate them into ten piles, based on the last digit of the ID. You then combine the piles in order, and separate them again into piles, based on the second to last digit of the ID. Do this five times.

5. The following C++ function, given a positive number $n$, returns $2n$. What is its loop invariant?

```
 1 int double(int n)
 2  {
 3   // input condition: n > 0
 4   int m = n;
 5   int rslt = 0;
 6   // loop invariant holds here
 7   while(m > 0)
 8    {
 9     // if the loop invariant holds here
10     m = m-1;
11     rslt = rslt + 2;
12     // it holds here
13    }
14   // loop invariant holds here
15   return rslt;
16  }
```

$m \geq 0$ and $2m + \text{rslt} = 2n$

6. The following function computes the product of two non-negative integers. Verify that that `p + cd = ab` is a loop invariant for this code, and use the loop invariant to show that `product(a,b)` returns `ab`.

```
 1 product(int a, int b)
 2  {
 3   assert (a >= 0 and b >= 0);
 4   int c = a;
 5   int d = b;
 6   int p = 0;
 7   // Verify that the loop invariant holds here.
 8   while(d > 0)
 9    {
10     // If the loop invariant holds here:
11     if(d % 2) p = p+c;
12     c = c+c;
13     d = d/2; // truncated division
14     // then the loop invariant holds here.
15    }
16   // It follows that the loop invariant holds here.
17   // We conclude that p = ab.
18   return p;
19  }
```

The loop invariant is $d \geq 0$ and $p + cd = ab$. It holds at line 7 because $c = a$, $d = b$, and $p = 0$. For the inductive step, assume the invariant holds at the beginning of an iteration, at line 10. Let $p'$, $c'$, $d'$ be the values of $p$, $c$, and $d$ at the end of the iteration, at line 14. $c' = 2c$. If $d$ is even, then $p' = p$ and $d' = \frac{1}{2}d$. If $d$ is odd, then $p' = p + c$ and $d' = \frac{1}{2}(d-1)$.

We first prove $d' \geq 0$. If $d$ is even, then $d \geq 0$ and $d' = \frac{1}{2}d \geq 0$. If $d$ is odd, since the loop condition is false and $d > 0$ is an integer, $d \geq 1$, hence $d - 1 \geq 0$, and $d' - \frac{1}{2}(d-1) \geq 0$.

We now prove $p' + c'd' = ab$. If $d$ is even, then $p' + c'd' = p + (2c)(\frac{1}{2}d) = p + cd = ab$. If $d$ is odd, then $p' + c'd' = p + c + (2c)(\frac{1}{2}(d-1)) = p + c + c(d-1) = p + c + cd - c = p + cd = ab$.

After the loop is finished, $d >= 0$ by the invariant, and $d <= 0$, since the loop condition is false, hence $d = 0$. Since the loop invariant holds, $p = ab - cd = ab - 0 = ab$, which proves correctness.

7. The following function computes $x^b$ for a real number $x$ and a positive integer $b$. What is its loop invariant? Hint: Addition is to multiplication as multiplication is to exponentiation. The loop invariant is analogous to the loop invariant of the function `product` in the previous problem.

$x$ is analogous to $a$, $y$ is analogous to $c$, $z$ is analogous to $p$, while $b$ and $d$ are analogous to $b$ and $d$.

The loop invariant is $d \geq 0$ and $z * y^d = x^b$. The invariant holds at line 7 since $z * y^d = 1 * x^b$.

Let $d'$, $z'$, $y'$ be the values of the variables after one iteration. Assume the invariant holds at line 7.

At line 14, $y' = y^2$, and $d \geq 1$ since the loop condition is false. If $d$ is even, then $z' = z$ and $d' = \frac{1}{2}d > 0$. If $d$ is odd, then $z' = z * y$ and $d' = \frac{1}{2}(d-1) \geq 0$.

We now verify the inductive property. We have already shown $d' \geq 0$. If $d$ is even, $z' * (y')^d = z * (y^2)^(\frac{1}{2}d) = z * y^d = x^b$. If $d$ is odd, $z' * (y')^d = z * y * (y^2)^(\frac{1}{2}(d-1)) = z * y * y^(d-1) = z * y^d = x^b$.

Finally, at line 16 $z = z * 1 = z * (y)^0 = z * y^d = x^b$, hence the code is correct.

```
 1 float power(float x, int b)
 2  {
 3   assert(b > 0);
 4   float y = x;
 5   int d = b;
 6   float z = 1.0;
 7   // loop invariant holds here
 8   while(d > 0)
 9    {
10     // if invariant holds here
11     if(d % 2) z = z*y;
12     y = y*y;
13     d = d/2;
14     // then it holds here
15    }
16   // loop invariant holds here
17   return z;
18  }
```

8. In the handout `loopInv.pdf` a loop invariant is given for code of the Floyd Warshall algorithm. Explain why this loop invariant has the inductive property.

   Consider the code of the Floyd-Warshall algorithm. Let the vertices of a weighted directed graph $G$ be the integers $1, \ldots n$, and let $W[i, k]$ be the weight of the arc from $i$ to $k$, if such an arc exists. If not we let $W[i, k] = \infty$. In our code, $V[i, k]$ is the least weight of any path found, so far, from $i$ to $k$.

```
 1 For all i,k let V[i,k] = W[i,k] and back[i,k] = i.
 2 For all i let V[i,i] = 0.
 3 j = 0;
 4 // loop invariant holds here
 5 while(j < n)
 6  {
 7   // if the loop invariant holds here
 8   j++;
 9   for all i and k // really two nested loops
10     {
11      temp = V[i,j]+V[j,k];
12      if(temp < V[i,k])
13       {
14        V[i,k] = temp;
15        back[i,k] = back[j,k];
16       }
17     }
18   // then the loop invariant holds here
19  }
20  // the loop invariant holds here and j = n, hence V[i,k] is the shortest path from i to k.
```

The key to understand correctness of this code, which takes $\Theta(n^3)$ time, is to understand the loop invariant of the outer loop. The invariant contains nested quantifiers:

"For any $i$ and any $k$, $V[i,k]$ is the least weight of any path from $i$ to $k$ whose interior does not contain any vertex of index greater than $j$."

We need to prove that condition is a loop invariant. It is helpful to add arcs of weight $\infty$ between every $i$ and $k$ if there is no arc already. These arcs do not affect the computation.

For any $i$, $j$, $k$, let $\sigma_j[i,k]$ be the least weight path from $i$ to $k$ whose interior does not contain any vertex of index greater than $j$[1] and let $V_j[i,k]$ be the weight of $\sigma_j[i,k]$. The loop invariant is that $V[i,k] = V_j[i,k]$.

Before the first iteration of the loop, at line 4, $j = 0$, hence $\sigma_j[i,k]$ consists of a single arc (possibly a fictitious arc of infinite weight) from $i$ to $k$, *i.e.* has no interior point; thus the invariant holds vacuously.

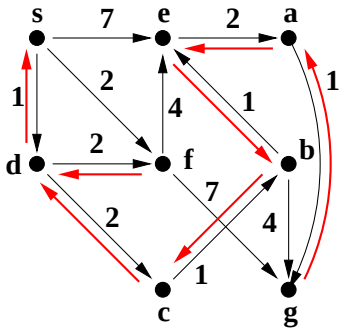
We need to prove the inductive property of the loop invariant. Suppose the invariant holds at the beginning of one iteration, at line 7. Let $j'$ be the value of $j$ at the end of the iteration, namely line 18. At line 7, $V[i,k] = V_j[i,k]$, the weight of $\sigma_j[i,k]$. Since the loop condition does not hold, $j < n$, hence $j' = j + 1 \leq n$, the first clause of the loop invariant at line 18. For the second clause, we need to prove that, at line 18, $V[i,k] = V_{j'}[i,k]$

Case I: $\sigma_{j'}[i,k]$ contains $j'$ as an interior point. Then $\sigma_{j'} = \alpha\beta$ (concatenation) Where $\alpha$ is a path from $i$ to $j'$ and $\beta$ is a path from $j'$ to $k$. Since $\alpha\beta$ is optimal, both $\alpha$ and $\beta$ must be optimal. By the loop invariant at line 7, neither $\alpha$ nor $\beta$ can contain $j'$, thus $\alpha = \sigma_j[i,j']$ and $\beta = \sigma_j[j',k]$. Thus $V_{j'}[i,k] = V_j[i,j'] + V_j[j',k]$, proving the second clause of the loop invariant.

Case II: $\sigma_{j'}[i,k]$ does not contains $j'$ as an interior point. In that case, $\sigma_{j'}[i,k] = \sigma_j[i,k]$, and $V[i,k]$ does not change, thus $V[i,k] = V_j[i,k] = V_{j'}[i,k]$, proving the second clause.

At line 20, $j = n$, and the loop invariant holds vacuously, because there is no vertex of index greater than $j$. Thus the code is correct.

9. Walk through Dijkstra's algorithm for the weighted directed graph $G$ shown below, where **s** is the start vertex.




| s | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 0 | 7 | 4 | 3 | 1 | ~~7~~ 5 | 2 | ~~9~~ 8 |
| * | e | c | d | s | ~~s~~ b | s | ~~f~~ b |

[1] That rule only applies to the interior vertices of $\sigma_j[i,k]$, not to the end points $i$ and $k$.

10. Find the strong components of the directed graph shown below, using the DFS algorithm shown in class and in your textbook. Show steps.