University of Nevada, Las Vegas Computer Science 477/677 Spring 2023

Study Guide for Third Examination April 12, 2023

1. Review answers to homework5:
   `http://web.cs.unlv.edu/larmore/Courses/CSC477/S23/Assignments/hw5ans.pdf`

2. Review answers to homework6:
   `http://web.cs.unlv.edu/larmore/Courses/CSC477/S23/Assignments/hw6ans.pdf`

3. True or False, Open if the answer is not known at this time.

   (i) **T** $\log^*(2^n) = \Theta(\log^*(n))$

   In fact, $\log^*(2^n) = \log^*(n) + 1$.

   (ii) **T** It is possible to execute the A^* algorithm on a weighted directed graph with some negative arcs, provided the heuristic satisfies the consistency condition.

4. Fill in the blanks.

   (a) **Dijkstra's** algorithm cannot be executed if any arc has negative weight.

   (b) No shortest path algorithm can be executed on a directed graph with a **negative cycle**. (Two words.)

   (c) The items in any priority queue represent **unfulfilled obligations**. (Two words.)

   (d) One sorting algorithm that we learned in class, **radix sort**, or **bucket sort**, does not use the comparison/exchange model of computation.

5. Using the stack algorithm, convert each infix expression to postfix. Show the stack at each step.

   For convenience, I show the stack horizontally, with the bottom on the left.

   (a) $a - b * c - d - e$

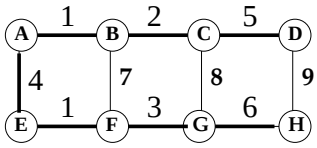
stack	input	output
	$a - b * c - d - e$	
	$-b * c - d - e$	a
$-$	$b * c - d - e$	a
$-$	$*c - d - e$	ab
$-*$	$c - d - e$	ab
$-*$	$-d - e$	abc
$-$	$-d - e$	$abc*$
	$-d - e$	$abc* -$
$-$	$d - e$	$abc* -$
$-$	$-e$	$abc* -d$
	$-e$	$abc* -d-$
$-$	e	$abc* -d-$
$-$		$abc* -d - e$
		$abc* -d - e-$

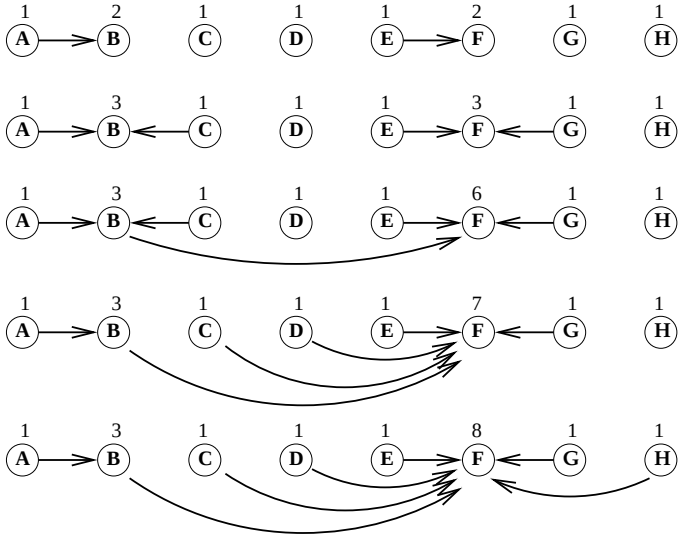
(b) In this example, we use the operator $\wedge$ (present in some languages, but not in C++) which is right-associative, and has higher precedence than multiplication and lower precedence than negation.

$a * b - d - e \wedge f \wedge -g$

stack	input	output
	$a * b - d - e \wedge f \wedge -g$	
	$*b - d - e \wedge f \wedge -g$	a
	$b - d - e \wedge f \wedge -g$	a
	$-d - e \wedge f \wedge -g$	ab
	$-d - e \wedge f \wedge -g$	$ab*$
$-$	$d - e \wedge f \wedge -g$	$ab*$
$-$	$-e \wedge f \wedge -g$	$ab * d$
	$-e \wedge f \wedge -g$	$ab * d-$
$-$	$e \wedge f \wedge -g$	$ab * d-$
$-$	$\wedge f \wedge -g$	$ab * d - e$
$-\wedge$	$f \wedge -g$	$ab * d - e$
$-\wedge$	$\wedge - g$	$ab * d - ef$
$- \wedge \wedge$	$-g$	$ab * d - ef$
$- \wedge \wedge \sim$	g	$ab * d - ef$
$- \wedge \wedge \sim$		$ab * d - efg$
$- \wedge \wedge$		$ab * d - efg \sim$
$- \wedge \wedge$		$ab * d - efg \sim$
$-\wedge$		$ab * d - efg \sim \wedge$
$-$		$ab * d - efg \sim \wedge\wedge$
		$ab * d - efg \sim \wedge \wedge -$

6. Walk through Kruskal's algorithm, using union/find, for the following weighted graph. Be sure to watch for path compression.





1. Find (A) = A. Find(B) = B
   Union(A,B) Select AB
2. Find(E) = E. Find(F) = F
   Union(E,F). Select EF.
3. Find(B) = B. Find(C) = C
   Union(B,C). Select BC
4. Find(F) = F. Find(G) = G.
   Union(F,G) Select FG
5. Find(A) = B. Find(E) = F.
   Union(B,F). Select AB
6. Find(C) = F. Path Compression. Find(D) = D. Union(C,D) Select CD.
7. Find(G) = F Find(H) = H Union(F,H

7. Given an ordered list of n numbers, how would you find the median number in $O(1)$ steps?

   The middle number in the list. (For this problem, don't worry about ties.)

8. Given two ordered lists of n numbers each, how would you find the median of the union of those two sets of numbers in $O(n \log n)$ time?

   Sort the numbers, pick the middle.

   Given two ordered lists of n numbers each, how would you find the median of the union of those two sets of numbers in $O(n)$ time?

   This solution was given in class by a student. Assume the lists are $x[1] \ldots x[n]$ and $y[1] \ldots y[n]$.

```
i = 1
j = 1
while(i+j < n)
  if(x[i] < y[j]) i++
  else j++
                Pick x[i] or y[j], whichever is larger.
```

   Given two ordered lists of n numbers each, how would you find the median of the union of those two sets of numbers in $O(\log n)$ time?

   I know of two solutions, one using recursion, the other using binary search.

   Here is the recursive solution. If the median of the first list is less than the median of the second list, discard the first half of the first list and the second half of the second list. Otherwise, discard the second half of the first list and the first half of the second list. In either case, you will now have two sorted lists of half the size. After doing this $\Theta(\log n)$ time, you will be left with two numbers: pick either one.

   Here is the other solution. Use binary search to select the largest index i such that $x[i] \leq y[n-i]$ Then pick $x[i]$ to be your median.

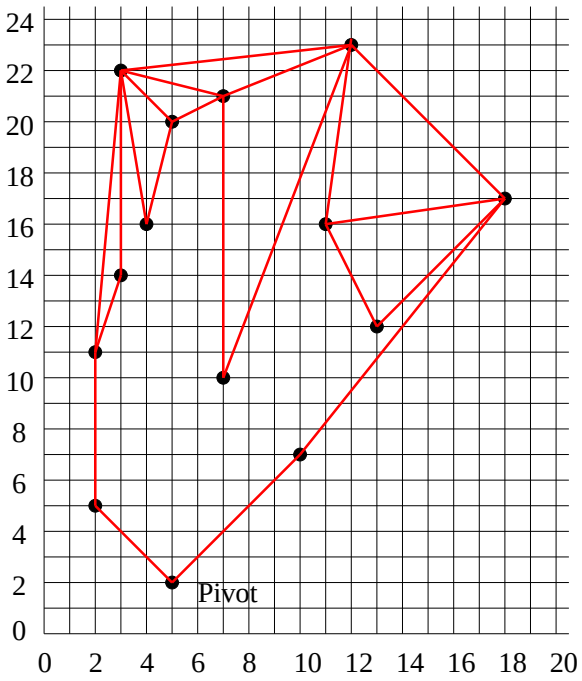
9. If you need to find the median of a list of three numbers using the comparison/exchange model of computation, the fastest method is to bubblesort the list in 3 comparisons, then pick the middle item.

   But what if you want the median of five numbers? How many comparisons do you need? (This problem arises in the BFPRT (median of medians) algorithm for median finding.)

   I won't ask this question on the exam, and I will show you how to do this in class afterwards.

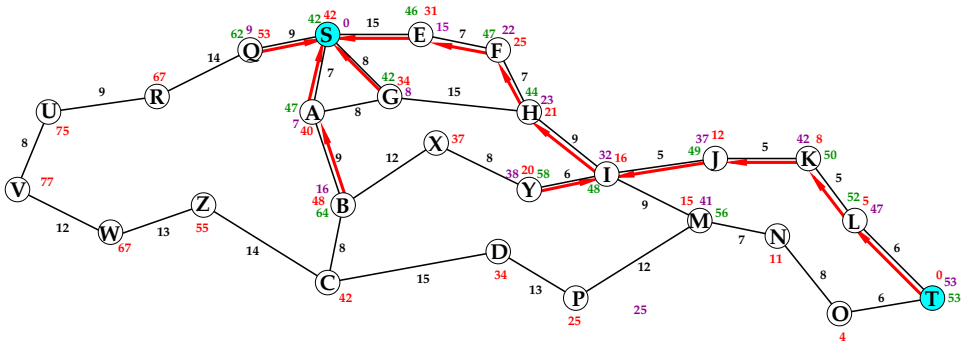
10. The convex hull of a set of n points in the plane can be found in $O(n)$ time using the algorithm *Graham Scan*, by Ron Graham. Walk through Graham Scan for the set of points shown in the figure below. As you draw lines, do not delete previously drawn lines.

    I used the lowest point as the pivot. If you picked a different pivot, your graph could look different.



11. Walk through the A^* algorithm for the weighted graph below, where S and T are the source and target vertices.

    f is shown in magenta and g in green. The backpointers are red arrows.



12. (a) You are given a set of n integers $x_1, \ldots x_n$ and an integer K. Write an $O(nK)$-time dynamic programming algorithm which determines whether some subset of the integers has total K.

C++ code for the algorithm:

```
bool s[n+1][K+1]; // defaults to false
s[0][0] = true;
 for(int i = 1; i <= n; i++)
  for(int k = 0; k <= K; k++)
   {
    s[i][k] = s[i-1][k];
    if(x[i]+k <= K)
     s[i][x[i]+k] = s[i-1][k];
   }
if(s[n][K]) cout << "There is a solution";
else cout << "There is no solution";
cout << endl;
```

(b) Answer the following question. "You know that this is the the subset sum problem, which you have learned is $\mathcal{NP}$–complete. Yet, I am asking you to find a polynomial time algorithm for that problem. How can that be?"

Time complexity is measured as a function of the number of bits of input, not the size of the input numbers. For example, if each of the inputs has n bits, the number of columns of the matrix is on the order of 2^n, which is an exponential function of the number of bits of the input. Thus, the algorithm is not truly polynomial time.