## University of Nevada, Las Vegas Computer Science 477/677 Fall 2020

## Answers to Assignment 5: Due Thursday October 1, 2020

**Name:**_______________________________________________________________

You are permitted to work in groups, get help from others, read books, and use the internet. Your answers must be written in a pdf file and emailed to the graudate assistant, Tandreana Chua `chuat4@unlv.nevada.edu` , by midnight October 1. Your file must not exceed 5 megabytes, and must print out to at most 4 pages.

All answers to the first two questions will be found in the following list of functions: $\sqrt{n}$, $n\sqrt{n}$, $n$, $n^2$, $n^3$, $n^4$, $\log n$, $\log\log n$, $\log^2 n$, $n \log n$, $n^2 \log n$.

1. Give an asymptotoic time complexity to each of these code fragments. The answer should be expressed using $\Theta$ notation, except for (1j).

   Instead of just guessing, or asking for the answer from someone, first make a serious effort to solve them yourself using the techniques I've shown you. In some cases, it might help to actually implement the code. Here is a suggestion on how you could do that.

```
cout << "Enter a positive integer: ";
cin >> n;
cout << endl;
int kount = 0;
for(int i = 0; i < n; i++)
  {
   kount++;
  }
cout << "for n = " << n << " kount = " << kount << endl;
```

(a)
```
for(int i = 0; i < n; i++)
  for(int j = 0; j < n; j++)
```
$\Theta(n^2)$

(b)
```
for(int i = 1; i < n; i = 2*i)
```
$\Theta(\log n)$

(c)
```
for(int i = 1; i < n; i++)
  for(int j = n; j > 0; j = j/2)
```
$\Theta(n \log n)$

(d)
```
for(int i = 0; i < n; i++)
  for(j = i; j > 1; j = j/2)
```
$\Theta(n \log n)$

(e)
```
for(int i = 0; i < n; i++)
  for(j = n; j > i; j = j/2)
```
$\Theta(n)$

(f)
```
for(int i = n; i > 0; i = i/2)
  for(int j = i; j > 0; j = j/2)
```
$\Theta(\log^2 n)$

(g)
```
for(int i = 0; i < n; i++)
  for(int j = 0; j < i*i; j++)
    if(j%n == 0)
      for(int k = 0; k < j; k++)
```
$\Theta(n^4)$

(h)
```
for(int i = 2; i < n; i = i*i)
```
$\Theta(\log \log n)$

(i)
```
for(int i = 0; i < n; i++)
  for(int j = 2; j*j < i; j++)
```
$\Theta(n^{3/2}) = \Theta(n\sqrt{n})$

(j) The next problem cannot be given an answer using $\Theta$. There are two answers you must give; an upper bound using $O$ notation, and a lower bound using $\Omega$ notation. The two functions are different.

```
for(int i = 2; i < n; i = i*i)
  for(int j = 0; j < i; j++)
```
$\Omega(\sqrt{n})$ and $O(n)$

2. For each of these recursive functions, let T(n) be the time complexity of the function with input $n$. (n each case, write a recurrence for T(n), and then solve that recurrence.

(k)
```
void f(int n)
 {
  if(n > 0)
   {
    for(int i = 1; i < n; i++)
     for(int j = 1; j < n; j++)
      cout << "hello!" << endl
    f(n/2); f(n/2);
   }
 }
```
$T(n) = 2T(n/2) + n^2$

$T(n) = \Theta(n^2)$

(l)
```
void g(int n)
 {
  if(n > 0)
   {
    for(int i = 1; i < n; i++)
     for(int j = 1; j < n; j++)
      cout << "hello!" << endl
    g(n/2); g(n/2); g(n/2); g(n/2);
   }
 }
```
$T(n) = 4T(n/2) + n^2$

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

(m)
```
void h(int n)
 {
  if(n > 0)
   {
    for(int i = 1; i < n; i++)
     for(int j = 1; j < n; j++)
      cout << "hello!" << endl
    h(n/2); h(n/2); h(n/2); h(n/2); h(n/2); h(n/2); h(n/2); h(n/2);
   }
 }
```
$T(n) = 8T(n/2) + n^2$

$T(n) = \Theta(n^3)$

3. Walk through heapsort, using the method shown in class, for the array: **A Q R B X S M L N T**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| A | Q | R | B | X | S | M | L | N | T | initial array |
| A | Q | R | N | X | S | M | L | B | T | bubbledown 4 |
| A | Q | S | N | X | R | M | L | B | T | bubbledown 3 |
| A | X | S | N | Q | R | M | L | B | T | bubbledown 2 |
| A | X | S | N | T | R | M | L | B | Q | bubbledown 5 |
| X | A | S | N | T | R | M | L | B | Q | bubbledown 1 |
| X | T | S | N | A | R | M | L | B | Q | bubbledown 2 |
| X | T | S | N | Q | R | M | L | B | A | bubbledown 5: heapification complete |
| A | T | S | N | Q | R | M | L | B | **X** | main loop: exchange first with last in queue |
| T | A | S | N | Q | R | M | L | B | **X** | bubbledown 1 |
| T | Q | S | N | A | R | M | L | B | **X** | bubbledown 2: heap restored |
| B | Q | S | N | A | R | M | L | **T** | **X** | exchange first with last in queue |
| S | Q | B | N | A | R | M | L | **T** | **X** | bubbledown 1 |
| S | Q | R | N | A | B | M | L | **T** | **X** | bubbledown 3: heap restored |
| L | Q | R | N | A | B | M | **S** | **T** | **X** | exchange first with last in queue |
| R | Q | L | N | A | B | M | **S** | **T** | **X** | bubbledown 1 |
| R | Q | M | N | A | B | L | **S** | **T** | **X** | bubbledown 3: heap restored |
| L | Q | M | N | A | B | **R** | **S** | **T** | **X** | exchange first with last in queue |
| Q | L | M | N | A | B | **R** | **S** | **T** | **X** | bubbledown 1 |
| Q | N | M | L | A | B | **R** | **S** | **T** | **X** | bubbledown 2; heap restored |
| B | N | M | L | A | **Q** | **R** | **S** | **T** | **X** | exchange first with last in queue |
| N | B | M | L | A | **Q** | **R** | **S** | **T** | **X** | bubbledown 1 |
| N | L | M | B | A | **Q** | **R** | **S** | **T** | **X** | bubbledown 2: heap restored |
| A | L | M | B | **N** | **Q** | **R** | **S** | **T** | **X** | exchange first with last in queue |
| M | L | A | B | **N** | **Q** | **R** | **S** | **T** | **X** | bubbledown 1: heap restored |
| B | L | A | **M** | **N** | **Q** | **R** | **S** | **T** | **X** | exchange first with last in queue |
| L | B | A | **M** | **N** | **Q** | **R** | **S** | **T** | **X** | bubbledown 1: heap restored |
| A | B | **L** | **M** | **N** | **Q** | **R** | **S** | **T** | **X** | exchange first with last in queue |
| B | A | **L** | **M** | **N** | **Q** | **R** | **S** | **T** | **X** | bubbledown 1: heap restored |
| A | **B** | **L** | **M** | **N** | **Q** | **R** | **S** | **T** | **X** | exchange first with last in queue heap order restored |
| **A** | **B** | **L** | **M** | **N** | **Q** | **R** | **S** | **T** | **X** | exchange first with last in queue. Queue empty: done |

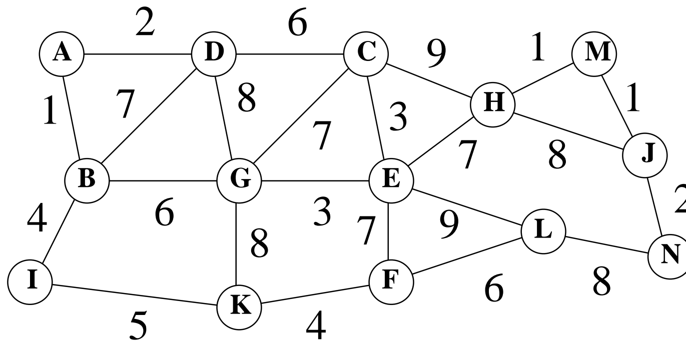
4. Walk through Kruskal's algorithm for a minimum spanning tree of the following weighted graph, showing the steps of union/find, and using path compression.

   There are many definitions of "*tree*." We use two of them in this problem:

   (a) A *tree* is a graph $T = (V, E)$ such that, for any $x, y \in V$, there is exactly one path through $T$ from $x$ to $y$.

   (b) A *tree* is a directed graph $T = (V, E)$ with a distinguished *root* $r \in V$ such that, for any $x \in V$, there is exactly one directed path from $x$ to $r$.

   A spanning tree is a tree in the sense of (a). Our data structure is a set of trees in the sense of (b).



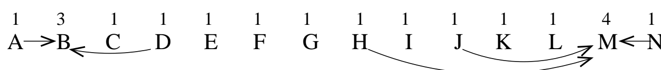

The edges sorted by weight:

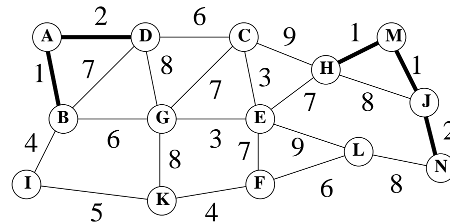
AB, HM, JM, AD, JN, CE, EG, BI, FK, IK, BG, CD, FL, BD, CG, EF, EH, DG, GK, HJ, LN, CH, EL

Kruskal's algorithm is defined below, where N is the number of vertices.
Initialize a forest of N singleton trees, each containig one vertex. The function Find(x) returns the leader of the set to which x belongs. If z and w are distinct leaders, Union(z,w) joins there sets together and chooses one of those to be the leader of the new set. The leader of the larger set becomes the leader of the union. If they are the same size, the larger leader becomes the new leader.

We process edges in weight order, selecting N-1 of them for the spanning tree and rejecting the others.
1. Repeat steps 2. through 4. until N-1 edges are selected.
2. Let {x,y} be the next edge.
3. Compute z = Find(x) and w = Find(y). (Path compression may occur here.)
4. If z and w are not equal, execute Union(z,w) and select the edge {x,y}.
5. Let T be the subgraph spanned by all selected edges. T is a minimal spanning tree.

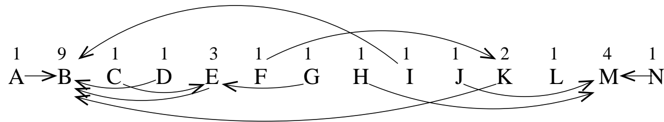
Process(A,B) Find(A) = A. Find(B) = B. Union(A,B). Leader[A] := B. Edge {A,B} selected.
Process(H,M) Find(H) = H. Find(M) = M. Union(H,M). Leader[H] := M. Edge {H,M} selected.
Process(J,M) Find(J) = J. Find(M) = M. Union(J,M). Leader[J] := M. Edge {J,M} selected.
Process(A,D) Find(A) = B. Find(D) = D. Union(B,D). Leader[D] := B. Edge {A,D} selected.
Process(J,N) Find(J) = M. Find(N) = N. Union(M,N); Leader[N] := M. Edge {J,N} selected.

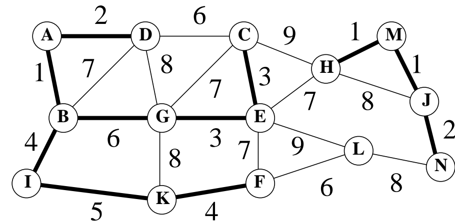



3

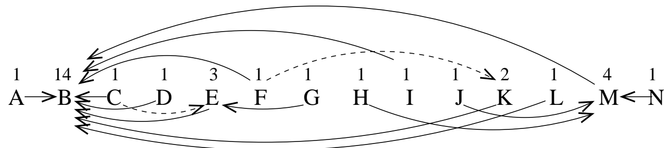
Process(C,E) Find(C) = C; Find(E) = E. Union(C,E). Leader[C] := E. Edge {C,E} selected.
Process(E,G) Find(E) = E; Find(G) = G. Union(C,G). Leader[G] := E. Edge {E,G} selected.
Process(B,I) Find(B) = B; Find(I) = I. Union(B,I). Leader[I] := B. Edge {B,I} selected.
Process(F,K) Find(F) = F; Find(K) = K. Union(F,K). Leader[F] := K. Edge {F,K} selected.
Process(I,K) Find(I) = B; Find(K) = K. Union(B,K). Leader[K] := B. Edge {I,K} selected.
Process(B,G) Find(B) = B; Find(G) = E. Union(B,E). Leader[E] := B. Edge {B,G} selected.
Process(C,D) Find(C) = B; Find(D) = B. No unionE

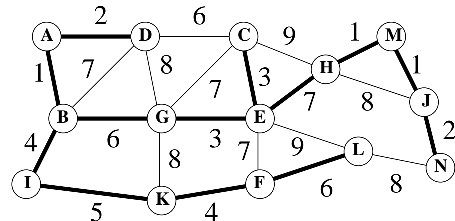



Process(F,L) Find(F) = B; Leader[F] := B (path compression); Find(L) = L. Union(B,L). Leader[L] = B. Edge {F,L} selected.
Process(B,D) Find(B) = B; Find(D) = B. No union.
Process(C,G) Find(C) = B; Leader[C] := B (path compression); Find(G) = B. No union.
Process(E,F) Find(E) = B; Find(F) = B; No union.
Process (E,H); Find(E) = B; Find(H) = M; Union(B,M); Leader[M] := B; Edge {E,H} selected.

Since N-1 = 13 edges have been selected, Kruskal's algorithm halts.



Below is code which implements Kruskal's algorithm. Let N be the number of nodes, which is 14 in our example problem. In the code, we assume, for simplicity, that the names of the nodes are 0 through 13.

```
int leader[N];// leader[x] is the leader of the set x belongs to, initially x
int parent[N];// initially undefined since every node is a leader
int size[N]; // size[x] is the number nodes in the set led by x, initially 1

int find(int x) // returns the leader of the set which contains x
 {
  if(leader[x] == x) return x;
  else
   {
    int p = parent[x];
    int ell = leader[p]; // recursive search for leader
    parent[x] = ell; // path compression
   }
  return ell;
 }

void union(int x, int y)
 {
  // input condition: x and y are different leaders
  if(size[x] < size[y] or size[x] == size[y] and x < y)
   {
    parent[x] = y;
    size[y] += size[x];
   }
  else
   {
    parent[y] = x;
    size[x] += size[y];
   }
 }

void processedge(int x, int y) // the next smallest edge is {x,y}
 {
  int z = find(x);
  int w = find(y);
  if(z != w)
   {
    union(z,w);
    cout << { << x << "," << y << "} is selected." << endl;
   }
 }
```