MIDTERM 3 Name: _____

CSCI 2270

Honor code: As always, I promise not to cheat or help others to cheat on this test.

_____

1a. (3 pts) Draw me the heap you get from adding the numbers 21, 34, 15, 33, 32, 47, 18, 47 to an empty heap. (Please draw these heaps in tree form.)

21 becomes the root

21

Add 34; swap with 21

34

21

Add 15: no problem

15

34

21

Add 33; swap with 21

15

34

33

21

Add 32: no problem

15

34

32

33

21

Add 47 and swap it with the 15 and the 34:

34

15

47

32

33

21

Add 18: no problem

18

34

15

47

32

33

21

Add 47 and swap it with 21 and then 33:

18

34

15

47

32

47

33

21

1b. (3 pts) Draw me the heap you get from adding the numbers 15, 18, 21, 32, 33, 34, 47, 47 to an empty heap.

For this question, every possible swap will happen.

15 becomes the root:

15

Add 18 and swap with 15:

18

15

Add 21 and swap with 18:

18

21

15

Add 32 and swap with 15 and 21:

18

32

21

15

Add 33 and swap with 21 and 32:

18

33

21

32

15

Add 34 and swap with 18 and 33:

33

18

34

21

32

15

Add 47 and swap with 33 and 34:

33

34

18

47

21

32

15

Add 47 and swap with 15 and 32:

33

34

18

47

21

47

32

15

Done.

1c. (4 pts) Draw me all of the 7 heaps you get as you remove the root element from the tree in 1b and reheapify, until the heap is completely empty. You don't have to draw the removed items.

Original heap:

33

34

18

47

21

47

32

15

Heap with 15 replacing the removed 47.

33

34

18

15

21

47

32

15 swaps with 47 and then 32. This is heap 1:

33

34

18

47

21

32

15

Remove 47, replace with 33:

34

18

33

21

32

15

Swap 33 with 34. This is heap 2:

33

18

34

21

32

15

Remove 34, replace with 18:

33

18

21

32

15

Swap 33 with 18. This is heap 3:

18

33

21

32

15

Remove 33, replace with 21:

18

21

32

15

Swap 21 with 32. This is heap 4:

18

32

21

15

Remove 33, replace with 15:

18

15

21

Swap 15 with 21. This is heap 5:

18

21

15

Remove 21, replace with 18. No swaps needed. This is heap 6:

18

15

Remove 18, replace with 15. No swaps needed. This is heap 7:

15

Remove 15, and heap is empty. Done.

2. (10 pts) Convert 13421 in base 5 to base 8, using the algorithm shown in class. Show your work.

(Note: my conversion constructor uses this algorithm.)

First, compute the digits from 0 to 5 in base 8. This is the easy part, since they're all the same:

Base 5 Base 8

0	0
1	1
2	2
3	3
4	4
5	5

Let's call m the number we want to obtain in base 8. Start m out as 0.

Loop over each digit in the old number,

m *= old base (in new base's arithmetic)

m += digit (in new base's arithmetic.)

Steps:

1. m *= old base; 0 * 5 = 0.
   m + 1 = 5.
   Double check: 1 in base 5 is 1 in base 8 (is 1 in base 10).
2. m *= old base; 1 * 5 = 5
   m + 3 = 10.
   Double check: 13 in base 5 is 10 in base (is 8 in base 10).
3. m *= old base; 10 * 5 = 50
   m + 4 = 54
   Double check: 134 in base 5 is 54 in base 8 (is 44 in base 10).
4. m *= old base; 54 * 5 = 334
   m + 2 = 336
   Double check: 1342 in base 5 is 336 in base 8 (is 222 in base 10).
5. m *= old base; 336 * 5 = 2126
   m + 1 = 2127
   Double check; 13421 in base 5 is 2127 in base 8 (is 1111 in base 10).

Done! 13421 in base 5 equals 2127 in base 8.

3. How many calls to pattern result from calling each of these? Count the original call as well as any others.

3a. (3 pts) pattern(cout, 256, 0);

= pattern(256)
+ 2 * pattern (128)
+ 4 * pattern(64)
+ 8 * pattern(32)
+ 16 * pattern(16)
+ 32 * pattern(8)
+ 64 * pattern(4)
+ 128 * pattern(2)

+ 256 * pattern(1)
(+ 512 * pattern(0))

_____

1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 = 511 (1023 if we count pattern(0) as a call)

3b. (3 pts) pattern(cout, 4096, 0);

= pattern(4096)
+ 2 * pattern (2048)
+ 4 * pattern(1024)
+ 8 * pattern(512)
+ 16 * pattern(256)
+ 32 * pattern(128)
+ 64 * pattern(64)
+ 128 * pattern(32)
+ 256 * pattern(16)
+ 512 * pattern(8)
+ 1024 * pattern(4)
+ 2048 * pattern(2)
+ 4096 * pattern(1)
(+ 8192 * pattern(0))

_____

1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512 + 1024 + 2048 + 4096 = 8193

(13683 if we count pattern(0) as a call)

3c. (3 pts) pattern(cout, n, 0); // assume n's a power of 2

= 2*n - 1 or 4*n - 1 if we count pattern(0) as a call

4. Bubble sort

4a. (3 pts) What's the best case run time for bubble sort?

O(n)

4b. (3 pts) What arrays produce this run time?

Arrays that are already sorted.

4c. (4 pts) What's the expected run time for bubble sort?

$O(n^2)$

5. (10 pts) Given the array 6 2 3 7 8 9 4 1 5 0, show me what it will look like after one round of partitioning (you can assume the simple partion, with no duplicates) in quicksort, assuming we have selected the 6 as the pivot. Showing your work here will help.

lt index starts at array[0]

gt index starts at array[9]

lt stops at array[3], the 7; gt stops at array[9], the 0; swap; 6 2 3 0 8 9 4 1 5 7

lt stops at array[4], the 8; gt stops at array[8], the 5; swap; 6 2 3 0 5 9 4 1 8 7

lt stops at array[5], the 9; gt stops at array[7], the 1; swap; 6 2 3 0 5 1 4 9 8 7

lt stops at array[7], the 9; gt stops at array[6], the 4; indexes have crossed;

swap arr[gt] and arr[0]; 4 2 3 0 5 1 6 9 8 7

Note that pivot is in the right place, and all numbers left of the pivot are less than the pivot, and all numbers right of the pivot are greater than the pivot.

6. (5 pts each) Tell me 2 important differences in big_number between what a copy constructor does and what an assignment operator (operator =) does. Justify your answers.

A copy constructor makes a new big_number a copy of an existing one, while assignment makes an existing big_number into a copy of an existing one.

1. This means that assignment must check for self assignment (big_number b; b = b;) but copy does not (big_number b(b); can't initialize a new b from an existing b if they are the same b.)

2. Assignment must also clear the existing digits from the list in order to avoid memory leaks. Constructors don't need to worry about this because they build a new number that has never had any digits before.

7. (10 pts) Memory bugs.

When operator = or operator += returns a big_number by reference, it's ok.

```
big-number& big_number::operator =(const big_number& m) {
        …
        return *this;
}
```

On the other hand, this code would crash…

```
big_number& no_no_nanette() {
        big_number answer = 9;
        return answer;
```

```
}
```

Why does the first function work fine, while the second one blows up?

Look at how the functions are used.

```
int main()

{
    big_number q(18);
    big_number r(10);
    q = r;                          // q still exists after this call; ok to return by reference in this case
    big_number s = no_no_nannette();    // the answer in no_no_nannette is destroyed
                                        // when the function returns, so we're assigning
                                        // s to a destroyed variable.  Bad things happen here.
```

8a. (5 pts) Tell me a good place to eat around Boulder. It can be cheap, fancy, or weird, as long as you like eating there better than you like eating paste.

8b. (5 pts) Tell me one good thing to order there.

My favorites; answers to 8a and 8b are both here.

The Med (good happy hour deals)

The polenta meatballs, or the chicken piccatta, or any tapa (steak with verde sauce is my current favorite)

Brasserie Tenten (also good happy hour deals, same owners as the Med):

Steak frites (with béarnaise), or, when in season, the tempura asparagus.

Parma in Louisville:

Try the duck prosciutto (unbelievable), or the prosciutto de Parma, or any of the 4 mozzarellas on the menu. It's hard to go wrong with this set of choices.