CMSC 15200 Final 3A

Alexandra Campili

TOTAL POINTS

69 / 100

QUESTION 1

1 Data Representation 14 / 16

- **0 pts** Correct

✓ **- 1 pts** **DR a) 3**

- **1 pts** Unsigned Min
- **1 pts** Unsigned Max
- **1 pts** Signed Min
- **1 pts** Signed Max
- **1 pts** Bitwise (1)
- **1 pts** Bitwise (2)
- **1 pts** Bitwise(3)
- **1 pts** Bitwise (4)
- **1 pts** Bitwise (5)
- **1 pts** Bitwise (6)

✓ **- 1 pts** **Bitwise (7)**

- **1 pts** DR a) 1
- **2 pts** DR a) 1
- **7 pts** Bitwise

QUESTION 2

2 Computational Complexity 3 / 10

- **0 pts** Correct

✓ **- 1 pts** **a) identify what variables influence execution time**

- **1 pts** a) circle the instructions that will be executed most
- **1 pts** a) define an equation for execution time

✓ **- 2 pts** **a) asymptotic complexity (big-O)**

- **1 pts** b) identify what variables influence execution time

✓ **- 1 pts** **b) circle the instructions that will be executed most**

✓ **- 1 pts** **b) define an equation for execution time**

✓ **- 2 pts** **b) asymptotic complexity (big-O)**

QUESTION 3

3 Heaps 20 / 20

✓ **- 0 pts** **Correct**

- **5 pts** a
- **5 pts** b
- **5 pts** c
- **5 pts** d

QUESTION 4

4 Heap Array-Based Implementation 18 / 20

- **0 pts** Correct
- **2 pts** Bubble should start at 1, not 0
- **6 pts** Not compare with parent

✓ **- 2 pts** **Not swapping the last element**

- **2 pts** Incorrect return
- **2 pts** Incorrect while loop
- **2 pts** Not decreasing num_filled
- **20 pts** Irrelevant to question
- **2 pts** no or incorrect boundary check
- **4 pts** the if statement should be in the while loop
- **16 pts** No bubbling or incorrect bubbling
- **2 pts** root is data[1]
- **8 pts** Incorrect bubbling for left and right separately
- **2 pts** Unnecessary swap during bubbling (swap only once on the same parent)
- **8 pts** Not swapping elements during bubbling
- **6 pts** Incorrect computation of children

QUESTION 5

5 Static variables implementation 12 / 25

- **0 pts** Correct
- **1 pts** undeclared variables
- **2 pts** no static char array

✓ **- 2 pts** **wrong buffer size**

- **3 pts** operator handling
- **3 pts** set operation

- **3 pts** append operation

✓ - **5 pts** **remove operation**

✓ - **6 pts** **insert operation**

- **2 pts** print result
- **3 pts** buffer not allocated properly
- **1 pts** overly complicated code
- **1 pts** out of bounds access
- **2 pts** buffer declaration / initialization
- **2 pts** memory leak
- **5 pts** use of undefined functions
- **3 pts** inexistent string library function names

QUESTION 6

6 Hash Tables 2 / 9

- **0 pts** Correct

✓ - **2 pts** **question (a)**

✓ - **2 pts** **question (b) change 1**

- **2 pts** question (b) change 2

✓ - **1 pts** **question (c) change 1**

✓ - **1 pts** **question (c) change 2**

✓ - **1 pts** **question (d)**

Your name: Alex Compiti

CS 152
Exam 3a
Winter 2019

Data Representation	/ 16
Algorithmic Complexity	/ 10
Heaps	/ 20
Array-based Implementation	/ 20
Static variables	/ 25
Hash Tables	/ 9
Total	/100

1. No one may leave for any reason and come back to work on his/her test. If you need to go to the restroom, go now.

2. You may have 1 sheet of notes, hand-written, double-sided

3. Your backpack must be zipped up.

4. You may not use any electronic devices of any kind. Make sure your cell phone is turned off so it does not ring during the test.

5. You may not wear hats, hoods, sunglasses, or do anything that would obscure your eyes.

6. You may not sit near your partner if you are partnered in programming or lab (within 3 seats in any direction)

8. We will not answer questions during the exam.

9. Do not hold your test up. It must stay on your desk.

10. If you seem to be looking around, you will be moved to the front to allow you to look around without having other students' test in your field of vision.

1. Data Representation (16 pts)

a) (5 pts) Each row represents the same number expressed in different formats. Fill in the blank spots.

Decimal	Unsigned binary (16 bits)	Signed 2's comp (16 bits)	Hex (4 digs)	Octal (6 digs)
34	0000 0000 0010 0010	X	0x0022	0o000042
-34	X	1111 1111 1101 1110	X	X
103	X	X	0x0037	X

b) (4 pts) If your computer represents integers in 11 bits, what are the ranges for integers? Express in decimal.

	MIN	MAX
unsigned	0	$2^{11}-1$
signed 2's complement	-2^{10}	$2^{10}-1$

Bitwise arithmetic – remember that bits are numbered from right to left, and numbering starts with 0.

c) (7 pts) For these questions, I have written a line of code in C or described what I want accomplished. Write one or a few lines of ***efficient*** C code that will accomplish the task. If you are supposed to set or extract a bit in a variable, do only that bit, not any in the rest of the number.

Code / description w/out bitwise ops	Code with bitwise ops
x = y / 64;	x = y >> 6 (shift y right 6 spots)
x = y % 16;	x = y & 15 (CLEARS ALL BITS TO LEFT OF FINAL FOUR; only keep first four bits)
Set bits 2 and 4 to 1 in variable a;	a = a | 20 (because 20 represented as 10100 - 2 & 4 set to 1)
x = y * 256;	x = y << 8
x = y % 32;	x = y & 31
Set bit 3 to 0 in variable a;	a = a & ~(1 << 3)

extract bit 7 in variable a and store into variable y	a = (a >> 7) & 1
	 if (a == 1) { y = y | (1 << 7); } // if a is 1, set bit 7 in y
	 if (a == 0) { y = y & ~(1 << 7); } // if a is 0, clear bit 7

2. Computational Complexity: (10 pts - 5 each)

Below are simplifications of actual algorithms used frequently in computer science:
For each problem:

a) Identify what variable(s) is the problems size (i.e. whose value(s) influence(s) the execution time)
b) Circle the instruction that will be executed the most times.
c) Create an equation to express how many times that instruction will be executed.
d) Express the computational complexity in big-O notation

a)

```
for (c = 0; c < m; c++) {
    for (d = 1; d < q; d*=2) {
        sum += d*c;
    }
}
```

→ this will be executed the most times

→ the outer loop is O(n)

→ the inner loop is O(log n)

first loop executed m times
second loop executes $\log_2(q)$

Problem size:
c, m, d, q
(c & d being incremented, m & q being bounds)

Equation:

$m \cdot \log_2(q)$

Complexity:
~~O(log n)~~
$O(n \log n)$

b)

```
void func(int *array, int n)        {
    if (n <= 1)
        return;
    func(array, n-1);
    int tmp = array[n-1];
    int i = n-1;
    while ((i > 0) && (tmp < array[i-1])){
        array[i] = array[i-1];
        i--;
    }
    array[i] = tmp;
}
```

→ O(n)

the second portion will never execute because it returns before it ever gets to the while loop, i.e. n keeps decreasing and once it gets to 1, the function stops running

Problem size:
n

Equation (show your work)

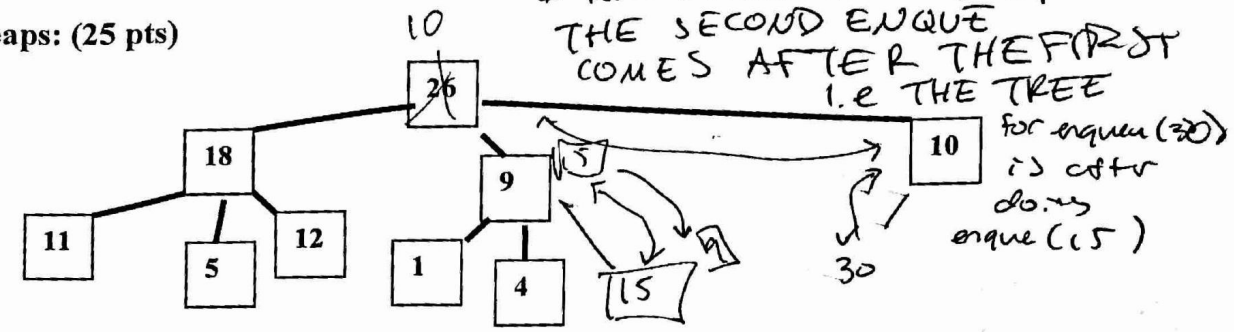
the recursive portion executed n times

the while loop never gets executed

$\sum_{i=0}^{-1} i$

Complexity:
O(1)

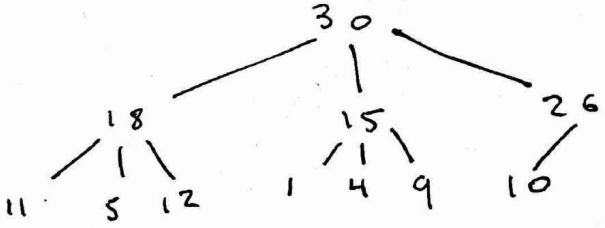
3. Heaps: (25 pts)



(5 pts each) Imagine that the heap is stored as a ternary tree in which every node has up to three children. Enqueue two numbers below. Draw the tree after each enqueue.

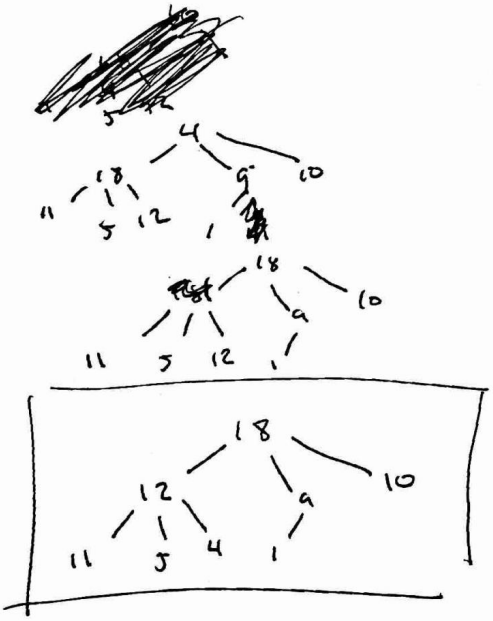
a) Enqueue(15)

b) Enqueue(30)

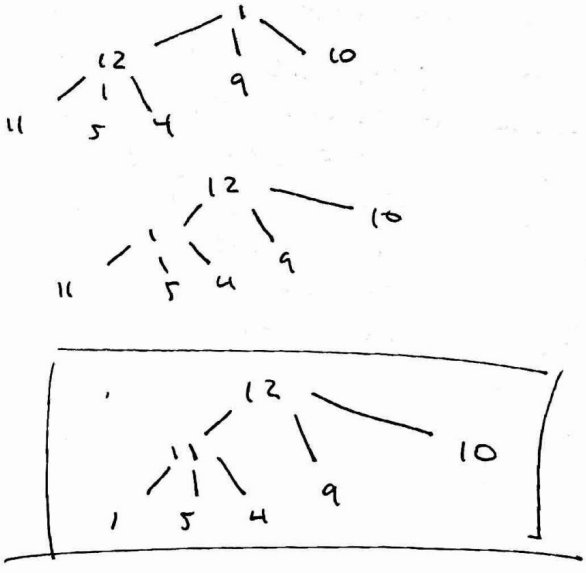


Now dequeue twice, but do so from the original tree above. Draw the tree after each dequeue

c) Dequene()



d) Dequeue()



6. Hash Tables (9 pts)

Provide the following explanations for how hash tables maintain O(1) performance.

a. (2 pts) What causes the hash table to make a major change?

If by make a change this means correcting for issues in the hash table, we say that collisions in hashing cause something "extra" to be done to the hash table / function & running out of slots in the table cause something extra to be done.

b. (4 pts) What two changes occur to the hash table?

Change 1: collision resolution (e.g. linear probing, quadratic probing, double hashing).

Change 2: resizing (increase or decrease the size of the hash table)

c. (4 pts) What root problem is each change solving? In other words, why do each of change 1 and change 2?

Change 1: ~~We need to~~ Since we have lots of large keys that are reduced to indices, it is possible that more than one key maps to the same place. Collision resolution allows us to find a new way to store the key if the slot it is mapped to is already taken

Change 2: We need to preserve O(1) access, but this can only happen if the # of operations on the table is proportional to the size of the table. Otherwise, insertions, deletions, and searches border on taking closer to O(n) time

d. (1 pt) Why do these changes cost a lot of time?

These changes require us to perform operations with linear run time. On average, (i.e. based on the frequency with which they are carried out) however, we obtain an amortized run time of O(1)

4. Heap Array-Based Implementation (20 pts)

Imagine you are implementing a function for an array-based heap. Each element in the array stores a pointer to the data. If there is no node for that particular location, then the pointer is set to NULL. Implement the dequeue function for a heap.

```
typedef unsigned int uint;
typedef struct {
        void **data;                    // an array of void pointers, each pointing to a data item
        uint num_allocated;             // length of the array
        uint num_filled;                // number of items currently in the heap
        int (*compare)(void *, void *); // comparison function for items in the array
} heap;
```

To access the data at the root: h->data[1];

* DOESN'T SAY IF THIS IS A MIN OR MAX HEAP - used min

```
void* dequeue(heap *h)
{
    if (h-> num_allocated <=0)
    { fprintf(stderr, "empty array\n"); }
    if (h-> num_allocated == 1)
    { h->num_filled--;
      min = h->data[1];
      h->data[1] = NULL;
      return min; }
    // extract min
    min = h->data[1];
    // set new data
    h->data[1] = h->data[h->num_allocated - 1];
    // decrease size (should be num_filled)
    h->num_filled--;
    // perform bubble
    bubble(h);
    // return pointer to dequed element
    return min;
}
```

```
// initialize to hold min
void *min;

void bubble(heap *h)
{ uint i=1
  uint tmp=1
  uint l, r;
  void *tmp2;
  while (i < h->num_filled)
  // start while loop
  { l=2*i+1;
    r=2*i+2;
    if (l < h->size && cmp(h->data[l], h->data[i]) == -1))
    { tmp=l; }
    if (r < h->size && cmp(h->data[r], h->data[i] == -1))
    { tmp=r; } set as minimum
    // now tmp is
    if (tmp != i)
    // if parent is not minimum, SWAP
    { tmp2 = h->data[i];
      h->data[i] = h->data[tmp];
      h->data[tmp] = tmp2;
    }
    i = tmp
  }
}
```

this is the bubble part

NOTES

```
void dequeue (heap)
{ //first set to null, decrease size of array
    h->data[i] = NULL;   tmp = h->data[i]
    h->num_allocated --;
    while called this
    if (h->num_allocated <= 0 )
        return ; //error
    if (h->num_allocated == 1 )
    { h->num_allocated --;
      return h->data[i];
    void *rt = h->data[1];
        h->data[1] = h->data[h->size -1];
        h->size --;
        HEAP BUBBLE
    { return h->data[1]

void bubble (heap *h)
  int i = 1      void* tmp2
  int l = 2*i+1
  int r = 2*i+2
  int tmp = l                cmp(      ) == -1
  if (l< h->size && h->data[l] < h->data[i])
     {tmp = l}
  if (r< h->size && h->data[r] < h->data[tmp])
        tmp = r
    if (tmp != i){
    { void* tmp2 = heap->data[i] ;
        heap->data[i] = heap->data[tmp] ;
        heap->data[tmp] = tmp2;
    bubble (tmp) ;} }
```

5. Static variables implementation (25 pts)

Static variables were an integral part of iterators. Now we will use them for a different purpose. You are writing an editor that stores one giant string – it can support 1024 characters. The editor can perform several actions, and each action modifies the string. There is no return value – the function **prints** the new string after each operation. Not all operations use all inputs. All operations use op. Set uses str, Insert uses str and start_index, Remove uses start_index and num_chars, and append uses str. You MAY use string library functions.

```
typedef unsigned int uint;
enum Operation { SET, INSERT, REMOVE, APPEND };
int calculate(enum Operation op, int rvalue);

int main()
{
        edit(SET, "How are you?");              // prints "How are you?"
        edit(APPEND, " Diana", 0, 0 );          // prints "How are you? Diana"
        edit(REMOVE, NULL, 11, 1);              // prints "How are you Diana"
        edit(INSERT, ",", 11, 0);               // prints "How are you, Diana"
}

void edit(enum Operation op, char *str, uint start_index, uint num_chars)
{
    // initialize static variable
    static char *out = NULL;
    // use switch statement to match
      operation to code
    switch (op) {
    case SET:
       out = strcpy(out, str);
       break;
    case APPEND:
       out = strcat(out, str);
       break;
    case REMOVE:
       out = strdel(out, start_index, num_chars);
       break;
    case INSERT:
       out = strinsert(out, start_index);
       break;

    // finally, print out the string
    printf("%s\n", out);

}
```

(not correct library name, but should insert at index in second parameter)