CS 477/677 Answers to Study Guide for Examination March 6, 2024

1. Fill in the blanks.

   (a) Any comparison-based sorting algorithm on a file of size n must execute $\Omega(n \log n)$ comparisons in the worst case. Use Ω.

   (b) Name two well-known divide-and-conquer sorting algorithms.

   mergesort

   quicksort

2. Fill in each blank. Write Θ if that is correct; otherwise write O or Ω, whichever is correct. Recall that log means $\log_2$.

   (a) $\log n^2 = O(\log n^3)$

   (b) $\log(n!) = \Theta(n \log n)$

   (c) $\sum_{i=0}^{n-1} i^k = \Omega(n^k)$

   (d) $n^n = \Omega(2^{\log^2 n})$.

   (e) $\log n = \Theta(\ln n)$

3. Fill in each blank with one of the words, *stack*, *heap*, *queue*, or *array*.

   (a) "pop" and "push" are operators of **stack**.

   (b) "fetch" and "store" are operators of **array**.

4. Find the asymptotic time complexity of each of these code fragments in terms of n, using Θ notation.

   (a)
   ```
   for(int i = 0; i*i < n; i++)
   ```
   $\Theta(\log \log n)$

   (b)
   ```
   for(int i = 0; i < n; i++)
    for(int j = 1; j < i; j = 2*j);
   ```
   $\Theta(n \log n)$

   (c)
   ```
   for(int i = 1; i < n; i++)
    for(int j = i; j < n; j = 2*j);
   ```
   $\Theta(n)$

   (d)
   ```
   for(float x = n; x > 2.0; x = sqrt(x))
   ```
   (`sqrt(x)` returns the square root of `x`.)

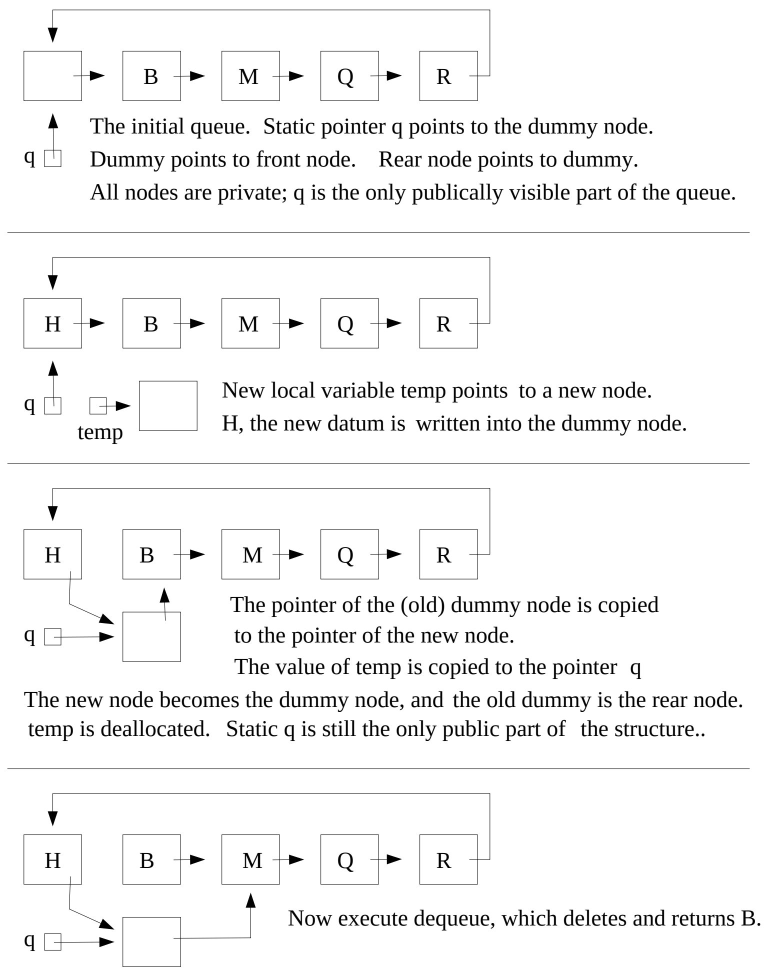
   $\Theta(\log \log n)$

(e) `for(int i = 1; i < n; i = 2*i)`
    `for(int j = 2; j < i; j = j*j);`

(Hint: use substitution)

$\Theta(\log n \log\log n)$

5. Show a circular queue with dummy node items B, M, Q, R, in that order, from front to rear. then show how the queue changes when you insert H.



6. A stack of integers could be implemented in C++ as a linked list as follows.

```
struct stack
 {
  int item;
  stack*link;
 };
```

Finish writing the code for the operators push, pop, and empty, below.

```
void push(stack*&s,int newitem)
 {
  stack*temp = new stack;
  temp->item = newitem;
  temp->link = s;
  s = temp;
 }

int pop(stack*&s)
 {
  int rslt = s->item;
  s = s->link;
  return rslt;
 }

bool empty(stack*s)
 {
  return s == NULL;
 }
```

7. Let $F_1, F_2, \ldots$ be the Fibonacci numbers. Find a constant K such that $F_n = \Theta(K^n)$. Show the steps.

   Assume $F_n = K^n$, although that's not exactly true. Then $K^n = K^{n-1} + K^{n-2}$ for each n. Divide by K^{n-2} and we get the quadratic equation $K^2 = K + 1$. By the quadratic formula $K = \dfrac{-1 \pm \sqrt{-1+4}}{2}$. But since the Fibonacci numbers are all positive, $K > 0$. Thus there is only one solution: $K = \dfrac{-1+\sqrt{3}}{2}$

8. (a) What is the purpose of the function `power` given below?

   To compute x^n.

   (b) Find a loop invariant of the while loop.

   $z * y^m = x^n$

```
float power(float x, int n) // input condition: n >= 0
 {
  int m = n;
  float y = x;
  float z = 1.0;
  while(m > 0)
   {
    if(m%2) z = z*y;
    m = m/2;
    y = y*y;
   }
  return z;
 }
```

9. The following code could be used in a C++ program implementing quicksort. What is the loop invariant of the loop?

```
void quicksort(int first,int last) // sorts the subarray A[first .. last]
 {
  if(first < last) // if first >= last, we are done
   {
    int mid = (first+last)/2;
    int pivot = A[mid];
    swap(A[mid],A[first]); // move pivot to first position
    int lo = first;
    int hi = last;
    while(lo < hi)
     {
      if(A[lo+1] > pivot and A[hi] < pivot)
       {
        swap(A[lo+1],A[hi]);
        lo++;
        hi--;
       }
      if(A[lo+1] <= pivot and lo < hi) lo++;
      if(A[hi] >= pivot and lo < hi) hi--;
     }
    assert(lo == hi);
    swap(A[first],A[lo]); // move pivot between subarrays
    if(lo < mid) // sort the smaller subarray first
     {
      quicksort(first,lo-1);
      quicksort(lo+1,last);
     }
    else
     {
      quicksort(lo+1,last);
      quicksort(first,lo-1);
     }
   }
 }
```

```
lo \le hi and A[first] = pivot and A[i] <= pivot for all first <= i <= lo
and A[i] >= pivot for all hi < i <= last
```

10. The following portion of C++ code contains an array implementation of `queue`. Fill in the missing code for the operators "`enqueue`" and "`empty`."

```
struct queue
 {
  int A[N]; // N is a constant large enough to prevent overflow
  int rear = 0;
  int front = 0; // initially the queue is empty
 };
void enqueue(queue&q,int newitem) // inserts newitem into q
 {
  q.rear++;
  q.A[q.rear] = newitem;
 }
bool empty(queue q) // returns true if q is empty, false otherwise
 {
  return q.rear == q.front;
 }
int dequeue(queue&q) // returns an item from q and deletes that item
 {
  int rslt = q.A[q.front];
  q.front++;
  return rslt;
 }
```

11. Fill in the blanks.

   (a) Name three search structures.
   binary search tree
   hash table
   unordered list

   (b) Name three priority queues.
   stack
   queue
   heap

   (c) Name a divide-and-conquer search algorithm, which only works on a sorted list.
   binary search

   (d) Name an $O(n)$-time search algorithm, generally used only when n is small.
   linear search

12. Solve the recurrences, expressing answers using Θ.

   (a) $F(n) = 2F(n/2) + n$
   $F(n) = \Theta(n \log n)$

(b) $F(n) = F(\sqrt{n}) + 1$

$F(n) = \Theta(\log \log n)$

13. Find the asymptotic time complexity of each of these code fragments in terms of n, using Θ notation.

(a)
```
for(int i = 0; i < n; i++)
 for(int j = 1; j < i; j = 2*j);
```
$\Theta(n \log n)$

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

(c)
```
for(float x = n; x > 2.0; x = sqrt(x))
```
(`sqrt(x)` returns the square root of `x`.)

$\Theta(\log \log n)$

(d)
```
for(int i = 1; i < n; i = 2*i)
 for(int j = 2; j < i; j = j*j);
```
(Hint: use substitution)

Substitute $k = \log i$, $m = \log n$, and $\ell = \log j$. The code becomes

```
for(int k = 0; k < m; k++)
 for(int l = 1; l < k; l = 2*l);
```

The solution is $\Theta(m \log m) = \Theta(\log n \log \log n)$

(e)
```
for(int i= 0; i < n; i++)
 for(int j = 0; j*j < n; j++)
```
The two loops are independent. The outer loop takes $\Theta(n)$ time, the inner $\Theta(\sqrt{n})$ time. Simply multiply the complexities, and we obtain $\Theta(n\sqrt{n}) = \Theta(n^{3/2})$.

(f)
```
for(float i = n; i >= 1.0; i = log(i))
```
We use the notorious $\log^*$ function. The answer is $\Theta(\log^* n)$.

(g)
```
for(int i = n; i > 1; i = i/2);
 for(int j = 1; j < i; j = 2*j);
```
Substituting $k = \log i$, $m = \log n$, $\ell = \log j$. we get

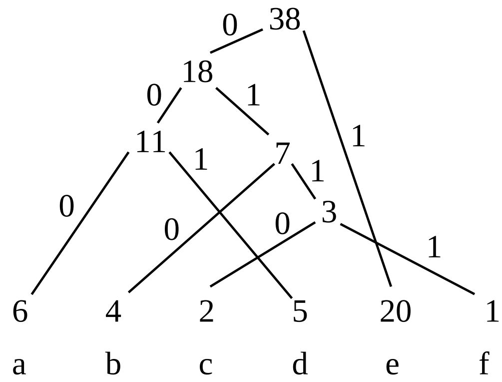
for$(k = m; k > 9; k - -)$

  for$(\ell = 0; \ell < k; \ell + +)$;

The complexity is $\Theta(m^2) = \Theta(\log^2 n)$.

14. Find an optimal prefix-free binary code for the following weighted alphabet. Show the Huffman tree.

a	6	000
b	4	010
c	2	0110
d	5	001
e	20	1
f	1	0111



15. Fill in the blanks.

(a) The items in a priority queue represent **unfulfilled obligations**

(b) Name three kinds of search structures. **binary search tree** **hash table** **unordered list**

16. Write the prefix expression equivalent to the infix epression $-a * b - (-c - d) \wedge e$

$- * \sim ab \wedge - \sim cde$

17. Walk through the stack algorithm to change the infix expression $-a + b \wedge c \wedge -f$ to postfix. Show the stack at each step.

stack	input	output	remarks
	$-a + b \wedge c \wedge -f$		
$\sim$	$a + b \wedge c \wedge -f$		Read, Push operator $\sim$
$\sim$	$+b \wedge c \wedge -f$	a	Read and write variable.
	$+b \wedge c \wedge -f$	$a \sim$	Pop and write $\sim$ since $+$ has lower priority
$+$	$b \wedge c \wedge -f$	$a \sim$	Read and push operator.
$+$	$\wedge c \wedge -f$	$a \sim b$	Read and write variable.
$+\wedge$	$c \wedge -f$	$a \sim b$	Read and push operator onto lower priority operator.
$+\wedge$	$\wedge - f$	$a \sim bc$	Read and write variable.
$+ \wedge \wedge$	$-f$	$a \sim bc$	Remember: $\wedge$ is right associative.
$+ \wedge \wedge \sim$	f	$a \sim bc$	$\sim$ has highest priority.
$+ \wedge \wedge \sim$		$a \sim bcf$	Read and write variable
$+ \wedge \wedge$		$a \sim bcf \sim$	Pop and write
$+\wedge$		$a \sim bcf \sim \wedge$	Pop and write
$+$		$a \sim bcf \sim \wedge\wedge$	Pop and write
		$a \sim bcf \sim \wedge \wedge +$	Pop and write
			Done.