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

Answers to Assignment 7: Due Saturday April 29, 2023

1. Fill in the blanks. For the first three of these questions, give the exact formula.

   (a) A binary tree of height h has at most $2^{h+1} - 1$ nodes.

   (b) A graph (not ordered graph) with n nodes has at most $\binom{n}{2}$, or $n(n-1)/2$. edges.

   (c) A planar graph with 2 nodes has at most 1 edge, while a planar grap with 3 nodes has at most 3 edges. If $n \geq 3$, a planar graph with n nodes has at most $3n - 6$ edges.

   For these questions, assume the graph has n vertices and m arcs.

   (d) The asymptotic complexity of the Floyd Warshall algorithm is $\Theta(n^3)$

   (e) The asymptotic complexity of the Bellman Ford algorithm is $O(nm)$.

   (f) The asymptotic complexity of Dijkstra's algorithm is $O(m \log n)$.

   (g) The asymptotic complexity of Johnson's algorithm is $O(nm \log n)$.

   For these questions, the number of items is n. Give the worst case complexity for each.

   (h) The asymptotic complexity of bubblesort is $O(n^2)$.

   (i) The asymptotic complexity of selection sort is $O(n^2)$.

   (j) The asymptotic complexity of tree sort is $O(n^2)$.

   (k) The asymptotic complexity of merge sort is $O(n \log n)$.

   (l) The asymptotic complexity of quicksort is $O(n^2)$.

   (m) The asymptotic complexity of polyphase merge sort is $O(n \log n)$.

   (n) The asymptotic complexity of the BFPRT algorithm is $\Theta(n)$.

2. Give the asymptotic complexity, in terms of `n`, of each of the following code fragments.

   (a)
   ```
   int kount = 0;
   for(int i = 2; i < n; i = i*i)
     kount++;
   cout << kount;
   ```
   Substitution. Let $j = \log i$ and $m = \log n$. We have:
   ```
   for(int j = 1; j < m; j = 2*j)
   ```
   Answer: $\Theta(\log m) = \Theta(\log \log n)$.

   (b)
   ```
   int kount = 0;
   for(int i = 1; i < n; i++)
     for(int j = 1; j < i; j = 2*j)
       kount++;
   cout << kount;
   ```
   Replace the inner loop by its known time complexity of $\Theta(\log n)$:

```
int kount = 0;
for(int i = 1; i < n; i++)
  kount = kount + log i
cout << kount;
```

Use integration, approximating the integer i by the real variable x, and approximating base 2 logarithm by the natural logarithm.

$$\text{kount} \approx \int_{x=1}^{n} \log x \, dx \approx \int_{x=1}^{n} \ln x \, dx = (x \ln x - x) \Big|_1^n = n \ln n - n - 1 \ln 1 + 1 = n \ln n - n + 1 = \Theta(n \log n)$$

(c)
```
int kount = 0;
for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j)
    kount++;
cout << kount;
```

By the same method as for the previous problem:

```
int kount = 0;
for(int i = 1; i < n; i++)
  kount = kount + log n - log i
cout << kount;
```

Again, using integration:

$$\text{kount} \approx \int_{x=i}^{n} (\ln n - \ln x) \, dx = (x \ln n - (x \ln x - x)) = n \ln n - n \ln n + n - 1 \Big|_1^n = \Theta(n)$$

(d)
```
int kount = 0;
for(int i = 1; i*i < n; i++)
  kount++;
```

Note that $i^2 < n$ is equivalent to $i < \sqrt{n}$.

```
int kount = 0;
for(int i = 1; i < sqrt{n}; i++)
  kount++;
```

kount $= \Theta(\sqrt{n})$.

(e)
```
int kount = 0;
for(int i = 1; i < n; i++)
  for(int j = n; j > i; j = j/2)
kount++;
```

The time complexity of the inner loop is $\Theta(\log n - \log i)$.

```
int kount = 0;
for(int i = 1; i < n; i++)
kount = kount + log n - log i
```

Using integration:

$$\text{kount} \approx \int_{x=1}^{n} (\ln n - \ln x)\, dx = (x \ln n - (x \ln x - x))\Big|_{x=1}^{n} = n \ln n - n \ln n + n - 1 = \Theta(n)$$

(f)
```
int kount = 0;
 for(int i = 1; i < n; i++)
   for(int j = i; j > 0; j = j/2)
kount++;
```

Replacing the inner loop

```
int kount = 0;
 for(int i = 1; i < n; i++)
   kount = kount + log i;
kount++;
```

$$\text{kount} \approx \int_{x=0}^{n} \ln x dx == (x \ln x - x)\Big|_{x=1} n = \Theta(n \log n)$$

3. What properties are desirable for a hash function h for a hash table used as a search structure?

   (a) It must be deterministic.

   (b) It must be quick to compute.

   (c) It must must be uncorrelated with any actual properties of the data.

4. The following code could be used as a subroutine for both quicksort and select. Assume A[n] is an array of integers. For simplicity, we assume that no two entries of A are equal. Write a loop invariant for the while loop.

```
int pivot = A[0];
int lo = 0;
int hi = n-1;
while(lo < hi)
 {
  if(A[lo+1] < pivot) lo++;
  else if(A[hi] > pivot) hi--;
  else swap(A[lo+1],A[hi]);
 }
```

`A[i]` $<$ `pivot` for any $0 < i \leq$ lo, and `A[i]` $>$ `pivot` for any hi $< i < n$

5. The main memory of your computer is probably a 1-dimensional array. That is, each variable of your program is stored in a location `RAM[i]` for some $0 \leq i < N$, where N is the size of your memory.

   Suppose you declare an array

   ```
   int A[20][100][40];
   ```

   Assume indices start at zero as in C++.

(a) If the compiler decides to store A in 80,000 consecutive locations, starting at `RAM[4000]`, in **row-major** order, where would `A[13][45][22]` be stored?

The offset is $13 * 100 * 40 + 45 * 40 + 22 = 53822$. The item is stored at `RAM[57822]`.

(b) On the other hand, suppose the compiler decides to store A in **column-major** order starting at `RAM[4000]`. In that case, where would `A[13][46][22]` be stored?

The offset is $20 * 100 * 22 + 20 * 46 + 13 = 44933$. The item is stored at `RAM[48933]`.

6. Explain how to use a search structure to implement a sparse array.

Let A be a sparse array. The search structure contains ordered pairs (memos) of the form (i, x). We call i the index and x the value of that pair, which indicates that `A[i] = x`. If there is no memo with index i, then $A[i]$ is the default value. The search structure has operators **insert** and **find**. where the key of each memo is its index.

To execute the assigment `store(i,x)` (that is, `A[i] = x`) execute `find(i)`. If there is a memo with index i, change the value that memo to x; otherwise, insert the memo (i, x). To obtain `fetch(i)`, that is return the value of `A[i]`, first execute `find(i)`. If there is a memo with index i, return the value of that memo; otherwise, return the default value.

7. Draw figures illustrating insertion (enqueue) into a queue implemented as singly linked circular list with dummy node.

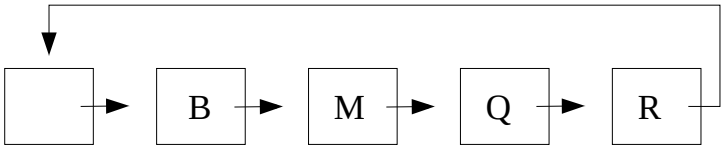
Start with a figure illustrating the structure when the items, from front to rear, are B, M, Q, R.

Next, show the steps needed to insert H.

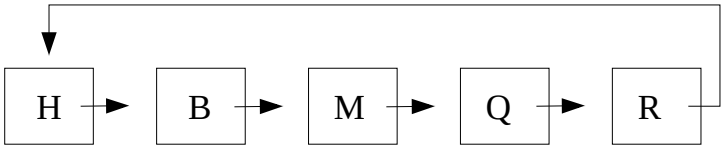
Here is code for the queue. For simplicity, I use **struct** instead of **class**.

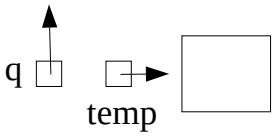
```
struct queuenode
 {
  char item; // or some other type
  queuenode*link;
 };

 void enqueue(queuenode*&q,char x)
  {
   queuenode*temp = new queuenode;
   q->item = x;
   temp->link = q->link;
   q->link = temp;
  }
```

This figure illustrates insertion of "H."

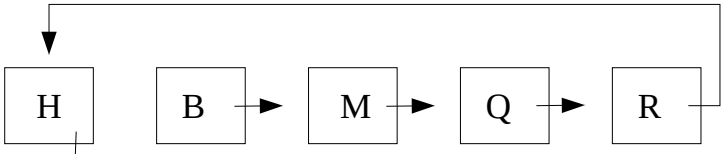


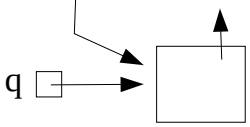
The initial queue. Static pointer q points to the dummy node.
Dummy points to front node. Rear node points to dummy.
All nodes are private; q is the only publically visible part of the queue.





New local variable temp points to a new node.
H, the new datum is written into the dummy node.





The pointer of the (old) dummy node is copied
to the pointer of the new node.
The value of temp is copied to the pointer q

The new node becomes the dummy node, and the old dummy is the rear node.
temp is deallocated. Static q is still the only public part of the structure..