Discussion #4 Outline

1. Announcements
   a. Homework/Project assignment and due dates
      i. Project 1 – Due Thurs. Oct. 7
      ii. Homework 2 – Due Thurs. Oct. 14

2. Pick up homework in hallway

3. Go over homework problems

4. Hashing Efficiency Review

5. Open Question Time
   a. Check for questions about Hashing, Project, Homework 2
Homework Coverage:

Problem #1
Quadratic Pseudocode
-----------------------------
Let A be an array of N numbers
for i<-0 to n-1 do
  a<-0
double a = 0;
for j<-0 to i do
  a<-a+X[j]
a = a + array[i];
A[i] <- a/(i+1)
averages[i] = a/(i+1);
return array A

Quadratic C++
--------------
double* averages;
for(int i = 0; i < size; i++)
  double a = 0;
for(int j = 0; j < i; j++)
  a<-a+X[j]
a = a + array[i];
averages[i] = a/(i+1);
return averages;

Linear Pseudocode
---------------------
Let A be an array of n numbers
for i<-0 to n-1 do
  s<-0
double s = 0;
for(int i = 0; i < size; i++)
  s<-s+X[i]
s = s + array[i];
A[i] <- s/(i+1)
averages[i] = s/(i+1);
return array A

Linear C++
----------
double* averages;
for(int i = 0; i < size; i++)
s = s + array[i];
averages[i] = s/(i+1);
return averages;

Problem #3
a) Convert the following algorithm from C++ to pseudocode using conventions described in the text.

```cpp
int lg (int n)
{
  for (int i = 0; n > 1; i++, n /= 2);
  return i;
}
```

Algorithm lg(n)
Input: integer n
Output: log_2(n)
i <- 0
while (n > 1) do
  i <- i + 1
  n <- n / 2
return i
b) Complete an operation count analysis of the pseudocode algorithm derived above. Include the cost of all primitive operations in your analysis (e.g., variable assignment, arithmetic operation, ...). That is, complete a ‘decomposition’ analysis of your algorithm.

1 assignment + (lg(n) + 1) comparisons + 2 lg(n) assignment + 2 lg(n) arithmetic + 1 return
= 5 lg(n) + 3 operations
arithmetic = 2 lg(n)
assignment = 2 lg(n) + 1
comparisons = lg (n) + 1
return = 1
Total = 5 lg(n) + 3

Problem #4

Derive and solve the recurrence relation for the following recursive algorithm.
int searchR(int a[], int v, int left, int right)
{
    if (left > right) return -1;
    int mid = (left + right)/2;
    if (v == a[mid]) return mid;
    if (v < a[mid])
        return searchR(a[], v, left, mid - 1);
    else
        return searchR(a[], v, mid + 1, right);
}

Recurrence Relation: T(n) = 1 + T(n/2), T(1) = 1, T(0) = 1

Solution:
T(n) = 1 + T(n/2)
assume n = 2^x (and x = lg n)
T(2^x) = 1 + T(2^(x-1))
= 1 + 1 + T(2^(x-2))
...
= x + T(2^0)
= x + 1
Plug in x = lg n
T(n) = lg n + 1
T(n) = O(lg n)
**Problem #5**

In class we described the implementation of list-based stack and an array-based queue. Below, show the code for an array-based stack and a list-based queue.

**Array-based stack**

```cpp
create(int size)
{
A = new T[size];
next = 0;
}

bool empty()
{
return (next == 0);
}

bool full()
{
return (next == size);
}

Push(T elt)
{
if (full())
throw exception;
A[next++] = elt;
}

T pop()
{
if (empty())
throw exception;
return A[--next];
}
```

**List-based queue**

```cpp
create()
{
head = NULL;
tail = NULL;
}

bool empty()
{
return (head == NULL);
}

bool full()
{
return false;
}

enqueue(T elt)
{
node *n = new node;
n->element = elt;
n->next = NULL;
if (tail)
tail->next = n;
tail = n;
if (head == NULL)
head = tail;
}

dqueue()
{
if (empty())
throw exception;
T result= head->element;
node *victim = head;
head = victim->next;
head = NULL;
tail = NULL;
}
```
delete victim;
}
if (head == NULL)
tail = NULL;
return result;

**Hashing Efficiency:**

a) In big-oh notation, how long does it take in the worst case to insert N keys into an initially empty hash table, using separate chaining with unordered, singly-linked lists?

$O(N)$

Reasoning: Insertion is O(1). No searching, just insert as head of linked list.

b) How long does it take in the worst case to insert N keys into an initially empty hash table, using separate chaining with sorted, singly-linked lists? (That is, any item inserted into the singly-linked list at any hash table address must maintain the sortedness of the list).

$O(N^2)$

Reasoning: Insertion is O(1). Searching: 1st key: 0; 2nd key: 1; 3rd key: 2; 4th key: 3; etc. Nth key: N-1; Average: (N-1)/2 = O(N)

$N$ inserts + $N$ searches = $O(N + N^2) = O(N^2)$

c) How long does it take in the worst case to insert N keys into an initially empty hash table, using linear probing?

$O(N^2)$

Reasoning: Insertion is O(1). Searching: 1st key: 0; 2nd key: 1; 3rd key: 2; 4th key: 3; etc. Nth key: N-1; Average: (N-1)/2 = O(N)

$N$ inserts + $N$ searches = $O(N + N^2) = O(N^2)$