CPSC 221: Theory assignment 1

Chin (Jean) Chen m4u9a 12714135
Ivan Zinin d9b0b 42108134

PART 1: INDUCTION

1) Let P(n) be the statement that 1^3 + 2^3 + … + n^3 = (n(n+1)/2)^2 for any positive integer n.

a) What is the statement P(1)?
b) Show that P(1) is true, completing the base case.
c) What is the inductive hypothesis?
d) What do you need to prove in the inductive step?
e) Complete the inductive step
f) Explain why the steps show that the formula is true whenever n is a positive int

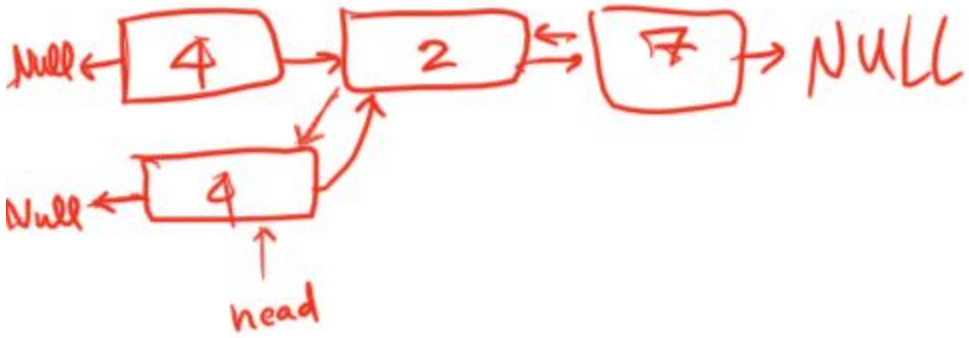
a) P(1) = (1(1+1)/2)^2 = 1
b) 1^3 = 1 = P(1)
c) Inductive hypothesis: Assume for an arbitrary integer k>0 that P(k) holds
d) Need to prove in the inductive step that if P(k) holds then P(k+1) holds too. I.e. 1^3 + … + (k+1)^3 = [(k+1)(k+2)/2]^2
e) From the inductive hypothesis, 1^3 + … + k^3 = (k(k+1)/2)^2
1^3 + … + k^3 + (k+1)^3 = (k(k+1)/2)^2 + (k+1)^3
= 0.25*[(k^2)(k+1)^2 + 4(k+1)^3]
= 0.25*{ (k+1)^2 * [k^2 + 4(k+1)] }
= 0.25*{ (k+1)^2 * (k+2)^2 }
= [(k+1)(k+2)/2]^2

f) The inductive steps show that the formula is true whenever n is a positive int because we have shown that P(1) is true; suppose k=1, then from the inductive step we know the k+1=2 case is true too, so P(2) is true. Then suppose k=2, then P(3) is true, and so on. There is no end in sight.

2) Prove that for all integers n>=1, 1+6+11+...+(5n-4) = n(5n-3)/2
Let P(n) = 1+6+11+...+(5n-4) for n integer, n >= 1,
P(1) = 1(5(1)-3)/2 = 1,
Assume P(k) = 1+6+11+...+(5k-4) = k(5k-3)/2,
Need to prove P(k+1) holds, ie: 1+6+11+...+(5(k+1)-4) = (k+1)(5(k+1)-3)/2
P(k+1) = 1+6+11+...+(5k-4)+(5(k+1)-4)
= k(5k-3)/2 + (5(k+1)-4)
= (5/2)k^2 -(3/2)k +5k +1
= (5/2)k^2 +(7/2)k+1
= (k+1)(5/2k +1) = (k+1)(5(k+1)-3)/2 = RHS
Thus P(n) holds.

PART 2: POINTERS

3)

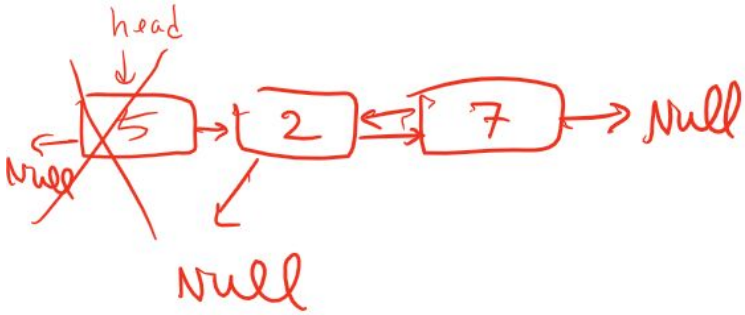


4)

```
// insert a node at the front of the list
Node *toAdd = new Node(4);
toAdd->next = head;
head->prev = toAdd;
toAdd->prev = NULL;
head = toAdd;
```

5)

6)



7) The above code is not safe because the statement “head = head->next” attempts to access the value pointed to by head, but the line above has already freed the memory for that element. The statement, therefore, is not guaranteed to set the head pointer correctly.

8)
//remove a node at the front of the list
head = head->next;
delete head->prev;
head->prev = NULL;

PART 3: ALGORITHM ANALYSIS

9) Find the smallest integer x such that f(n) is O(n^x) and positive constants c and n0 to satisfy the definition for Big-O defined in class, for each of the following functions:

a. f(n) = n^2 + 25n – 4
b. f(n) = 5n^3 + 3n^2 logn
c. f(n) = (n^4 + n^2 + 1) / (n^3 + 1)

If f(n) <= cn^x for all n >= n0

a) x = 2 c = 26 n0 = 1
n^2 + 25n - 4 <= n^2 + 25n^2 = 26n^2

b) x = 3 c = 8 n0 = 1
5n^3 + 3n^2 logn <= 5n^3 + 3n^3 = 8n^3 (since logn grows slower than n)

c) x = 1 c = 3 n0 = 1
(n^4 + n^2 + 1) / (n^3 + 1) <= (n^4 + n^4 + n^4) / (n^3) = 3n but when n<1 this does not hold so n0=1

10) Explain, by determining the expected run-time, the differences (if any) using an array-based versus a linked-list-based implementation for the following operations:
a. find(int index); // Returns the value at given index
b. remove(int index); // Removes the value at index and
// maintains adjacent ordering from
// start to finish (e.g., non- decreasing order).

a) The expected runtime for find() in an array is O(1). This is because any memory location in an array can be addressed instantaneously. Expected runtime for find() in a linked list

is O(n). This is because we need to iterate through the first n elements to find the element at the index.

b) The expected runtime for remove() in an array is O(n). This is because after removing the specified element, we need to shift the rest of the elements by one memory slot to the left. Expected runtime for remove() in a linked list is O(n). This is because we need to iterate through the elements until we found the one at the specified index and it can be “re-linked” in constant time.

11)) Give tight asymptotic worst-case bounds for the time complexity of each of the following pieces of C++ code. Show your work. How much time does each loop take? First calculate the running time, T(n), (which you can assume is the number of lines of code executed as a function of n) then express its order of growth using Θ-notation.

a. Let size represent n, the number of elements in the sorted array:

```
// Reverses the values in the array arr
void reverse(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        int temp = arr[0];
        for (int j = 0; j < size - i - 1; j++) {
            arr[j] = arr[j+1];
        }
        arr[size-i-1] = temp;
    }
}
```

a) Let n = size; T(n) <= 1+ n*5 + (1+1+1)[(n-1)+(n-2)+....+1]
= 1 + 5n + 3*\sum{i=1}^{n-1} i
= 1 + 5n + 3n(1+(n-1))*(n-1)/2
= 1 + 5n + 3n(n-1)/2
= (3/2)n^2 + (7/2)n + 1

Big O calculation:
T(n) <= cn^x for all n >= n0
x = 2 c = 3/2 n0 =1
(3/2)n^2 + (7/2)n + 1 <= (3/2)n^2 + (7/2)n^2 + (1)n^2

Big-omega calculation:
T(n) >= cn^x for all n >= n0
x = 2 c = 3/2 n0 =1
(3/2)n^2 + (7/2)n + 1 >= (3/2)n^2

Big-Omega (n^2) and Big-O (n^2) ⇒ Big-Theta (n^2)

b. Let size represent n, the number of elements in the sorted array:

```
// Searches a sorted array for the value n
// Returns index of n, or -1 if array does not contain n
int search (int arr[], int size, int n) {
    int min = 0, max = size-1;
    int mid;
    int max = size-1;
    while (min <= max) {
        mid = (min + max)/2;
        if (arr[mid] < n) {
            min = mid+1;
        } else if (arr[mid] > n) {
            max = mid-1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

T(n) <= (1+1+1+ 5(how many times the while loop runs-1) + (number of lines executed on the last time the loop runs, = 4) +1)

Let x = how many times the while loop runs-1,
Min is initially 0 and max is initially size-1 = n-1
Then since we are halving the interval between min and max each time we run the loop, the worst case is approximately n*(½)^x = 1 -> n = 2^x -> xln2 = ln(n) -> x = lg(n)

Thus,
T(n) <= 3 + (5 + 5 + 5 + 5…) + 4 + 1
= 5lg(n) + 8

Big O calculation:
T(n) <= c * lg(n) for all n >= n0
c = 13 n0 =1
5lg(n) + 8 <= 5lg(n)+ 8lg(n) = 13lg(n)

Big-omega calculation:
T(n) >= c * lg(n) for all n >= n0
c = 5 n0 =1
5lg(n) + 8 => 5lg(n)

Big-Omega (lgn) and Big-O (lgn) ⇒ Big-Theta (lgn)