**Problem 1:**

210 < 2logn < 3n + 100logn < 4n < nlogn < 4nlogn + 2n < n2 + 10n < n3 <2n

210 is constant = c

2logn = nlog2 < n (because log2 < 1)

3n + 100logn < 4n (because 100logn < n)

4n < nlogn (because 4 < logn)

nlogn < 4nlogn + 2n

4nlogn + 2n ~ nlogn but n2 + 10n ~ n2 so: 4nlogn + 2n < n2 + 10n

n2 + 10n < n3 < 2n

**Problem 2:**

*Pseudo-codes 1:*

LONG find2n (INT n)

{

LONG kq = 1;

FOR i = 1 TO i = n STEP i = i + 1 DO

kq = kq \* 2;

}

RETURN kq;

**Complexity O(n)**

*Pseudo-codes 2:*

LONG find2n (INT n)

{

LONG kq = 1;

LONG a = 2;

WHILE n > 0

{

IF n > 0

{

kq = kq \* 2;

--n;

}

a = a \* a;

n = n /2;

}

RETURN kq;

}

Complexity O(logn)

**Problem 3:**

***A Queue data structure includes operations:***

***- constructor***

***- size: return size of a queue***

***- dequeue: delete first element ~ the same as pop()***

***- enqueue(x): add x at tail of queue***

*Pseudo-codes 1:* Using an **array** with front of Queue is first element in Array

ARRAY Queue [...];

{

INT size = 0;

INT[] Queue;

}

INT size () {

RETURN size;

}

DEQUEUE () {

FOR i = 1 TO i = size STEP i = i + 1 DO:

{

Queue[i-1] = Queue[i];

}

size = size – 1;

}

*Complexity:* O(n) // with n ~ size

ENQUEUE (INT x)

{

size = size + 1;

Queue [size+1] = x;

}

*Complexity:* O(1)

***Complextiy:* O(n) + O(1) = O(n)**

**Problem 4:**

LINKEDLIST

*Pseudo-codes:* Using Node (Singly LinkedList) with head of list is front of queue

STRUCT NODE {

INT data;

NODE \*next;

}

INT size (NODE \* head) {

NODE\* temp = head;

INT p\_size = 0;

WHILE (temp -> next != NULL) {

p\_size++;

}

RETURN p\_size;

}

*Complexity* O(n) // with n ~ size

DEQUEUE (NODE\* head) {

NODE\* p = head;

head = head -> next;

DELETE temp;

RETURN head;

}

*Complexity* O(1)

ENQUEUE (NODE\* head, INT x) {

NODE\* p = head;

//goto the tail

WHILE (p -> next != NULL) {

p = p->next;

} // “size” time

size = size + 1;

NODE\* temp = NEW NODE (DATA = x);

temp -> next = NULL;

p -> next = temp;

RETURN head;

}

Complexity O(n) // with n ~ size

***Complexity* = O(1) + O(n) + O(n)= O(n)**

**Problem 5**

***A stack data structure includes operations:***

***- Constructor***

***- size: return size of stack***

***- top: return the value of top element***

***- pop: delete the top element of stack***

***- push(x): add x on the top of stack***

*Pseudo-codes:* Using an array (Top of stack is last of array)

ARRAY [] Stack {

INT [] Stack;

INT size = 0;

}

INT size {RETURN size;}

PUSH (INT x) {

size++;

Stack[size] = x;

}

INT top {RETURN Stack [size];}

POP { // Delete at last of array – top of stack

size = size – 1;

}

**Complexity: O(1)**

**Problem 6:**

LINKEDLIST

*Pseudo-codes:* Using **Node** (singly linkedlist), top of stack <- last of linkedlist

STRUCT Node{

INT data;

Node\* next;

}

INT size (Node\* head) {

Node\* p = head;

INT p\_size = 0;

WHILE (p -> next != NULL)

{

p\_size++;

p = p -> next;

}

RETURN p\_size;

}

PUSH (Node\* head, INT x) // insert at tail of linked list

{

Node\*p = head;

WHILE (p->next != NULL) {

p = p->next;

}

Node\*temp = new Node ();

temp->next = NULL;

temp->data = x;

p->next = temp;

RETURN head;

}

Complexity O(n)

POP (Node\* head) // remove at the tail of linked list

{

Node\* tail = head;

Node\* p = head;

WHILE (tail->next != NULL) {

tail = tail->next;

}

DELETE tail;

p->next = NULL;

RETURN head;

}

Complexity O(n)

***Complexity* = O(n) + O(n) + O(n) = O(n)**