COL106
Data Structures and Algorithms

Subodh Sharma and Rahul Garg

COL106 Plagiarism Policy and Honor Code

- Copying or collaborating or use of any other unethical means in exams, quizzes is not permitted
- Every assignment will have its own honor code
  - Indicating what is permitted and what is not
- Read the honor code carefully, ask in case of doubts
- Violation of Honor code will lead to
  - D grade
  - Disciplinary committee (DISCO)
  - Any other penalty deemed fit by the instructors

Should we have Quiz Today?

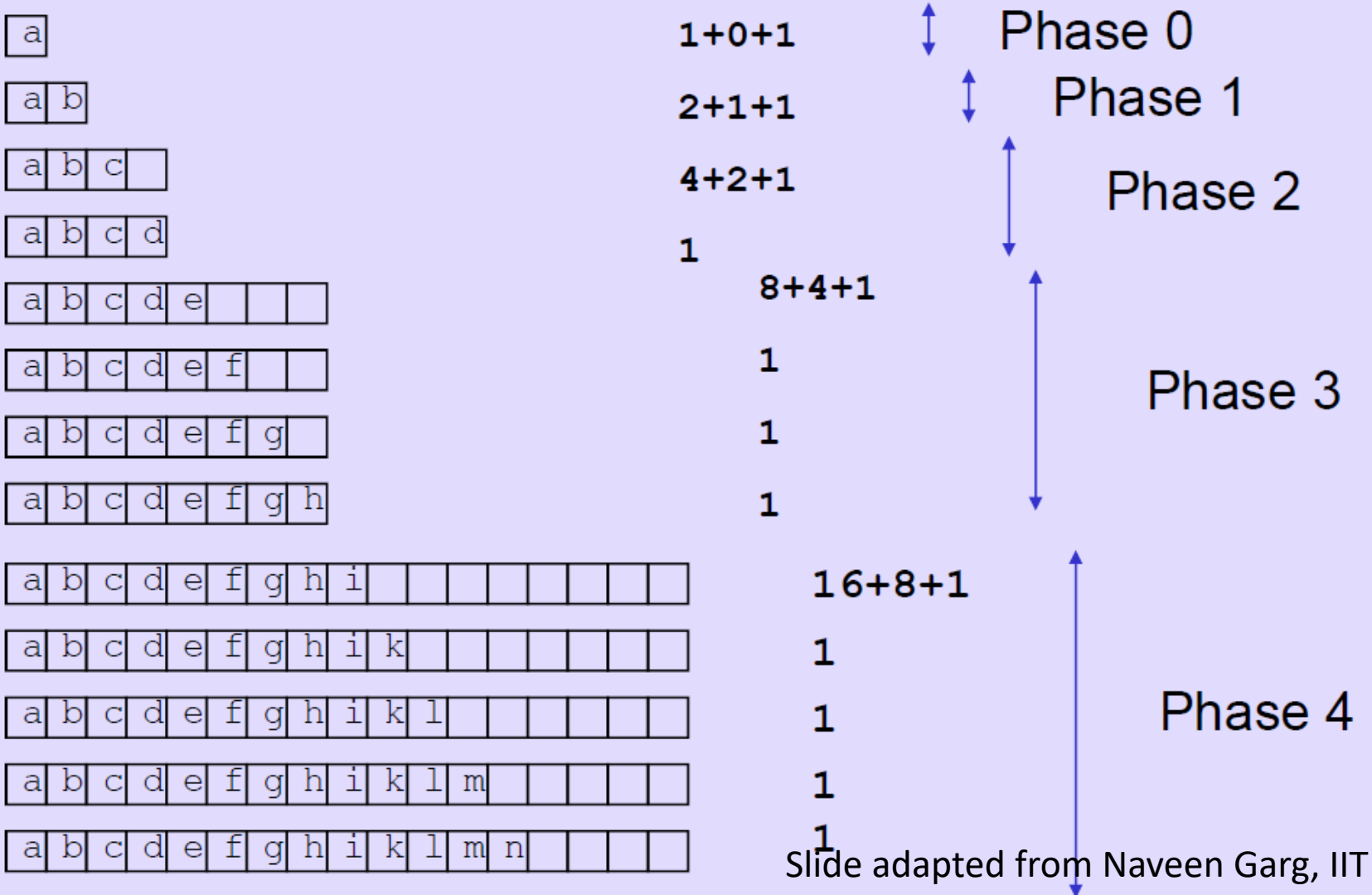
Dynamic Array-Based Stacks: Analysis

A Growable Stack: Multiplicative Increase

```
#define C 2
void Stack::push(int e) {
  if (t >= stackCapacity - 1) { // Reallocate space and copy
    int *temp = new int[stackCapacity * C]; // TBD: Check for error
    for (i = 0; i <= t; i++)
      temp[i] = S[i];
    delete S;
    S = temp;
    stackCapacity *= C;
  }
  S[++t] = e;
}
```

Analysis: Multiplicative Increase

start with an array of size 0. cost of a special push is 3N + 1



Slide adapted from Naveen Garg, IIT Delhi

Analysis: Multiplicative Increase

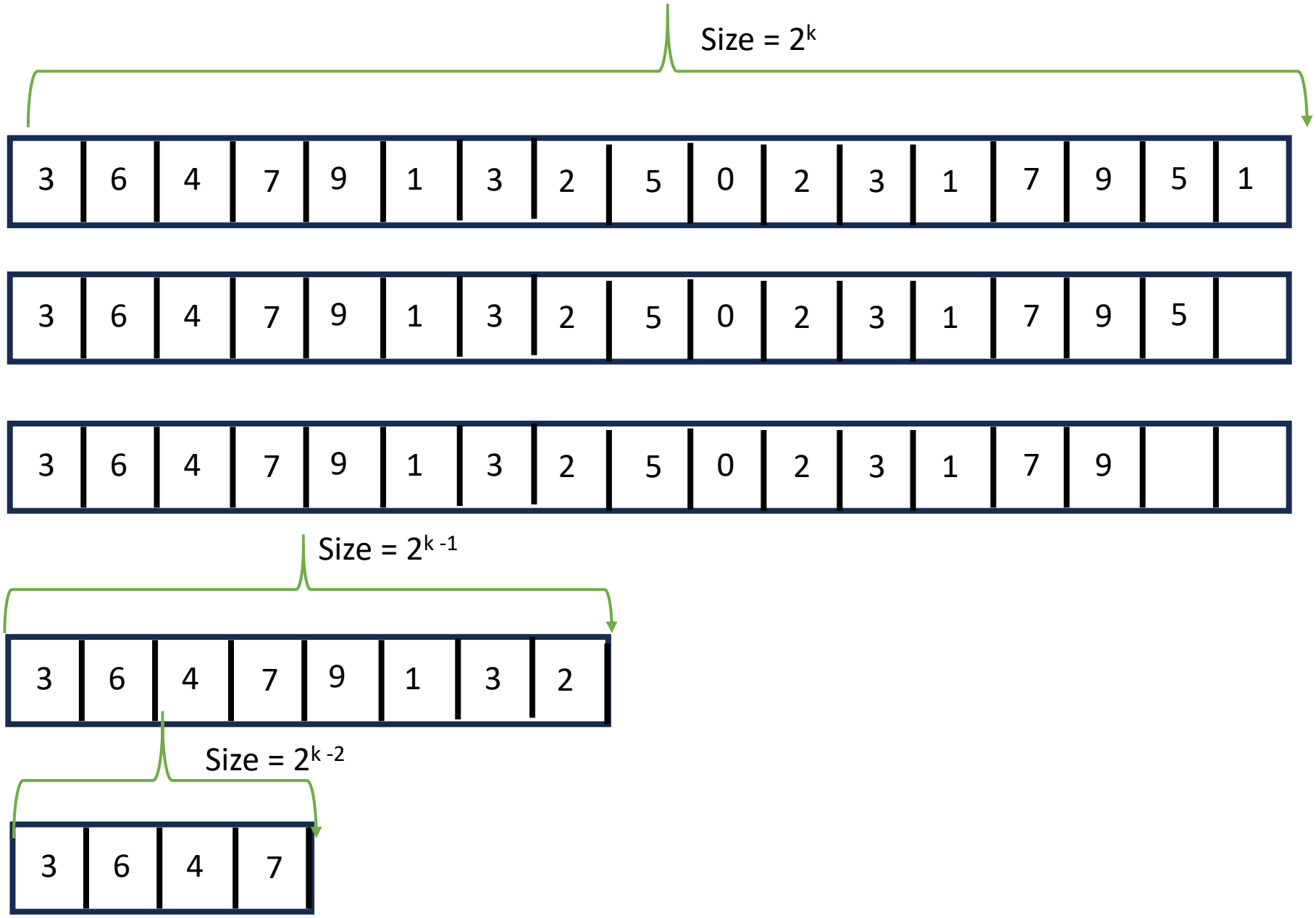
- In phase i the array has size 2^i
- Total cost of phase i is
  - 2^i is the cost of creating the array
  - $2^{i-1}$is the cost of copying elements into new array
  - $2^{i-1}$is the cost of the 2^{i-1}pushes done in this phase
- Hence, cost of phase i is 2^{i+1}
- If we do n pushes, we will have log n phases.
- Total cost of n pushes
- $= 2 + 4 + 8 + \ldots + 2^{\log n + 1} = 4n - 1$
- $T(n) = O(n)$

Slide adapted from Naveen Garg, IIT Delhi

Shrinking: Multiplicative Decrease

```
#define C 2
int Stack::pop() {
  int result = S[t];
  if (t <= stackCapacity / C) { // Reallocate space and copy
    int *temp = new int[stackCapacity / C]; // TBD: Check error
    for (i = 0; i <= t; i++)
      temp[i] = S[i];
    delete S;
    S = temp;
    stackCapacity /= C;
  }
  t--;
  return result;
}
```

Analysis: Multiplicative Decrease



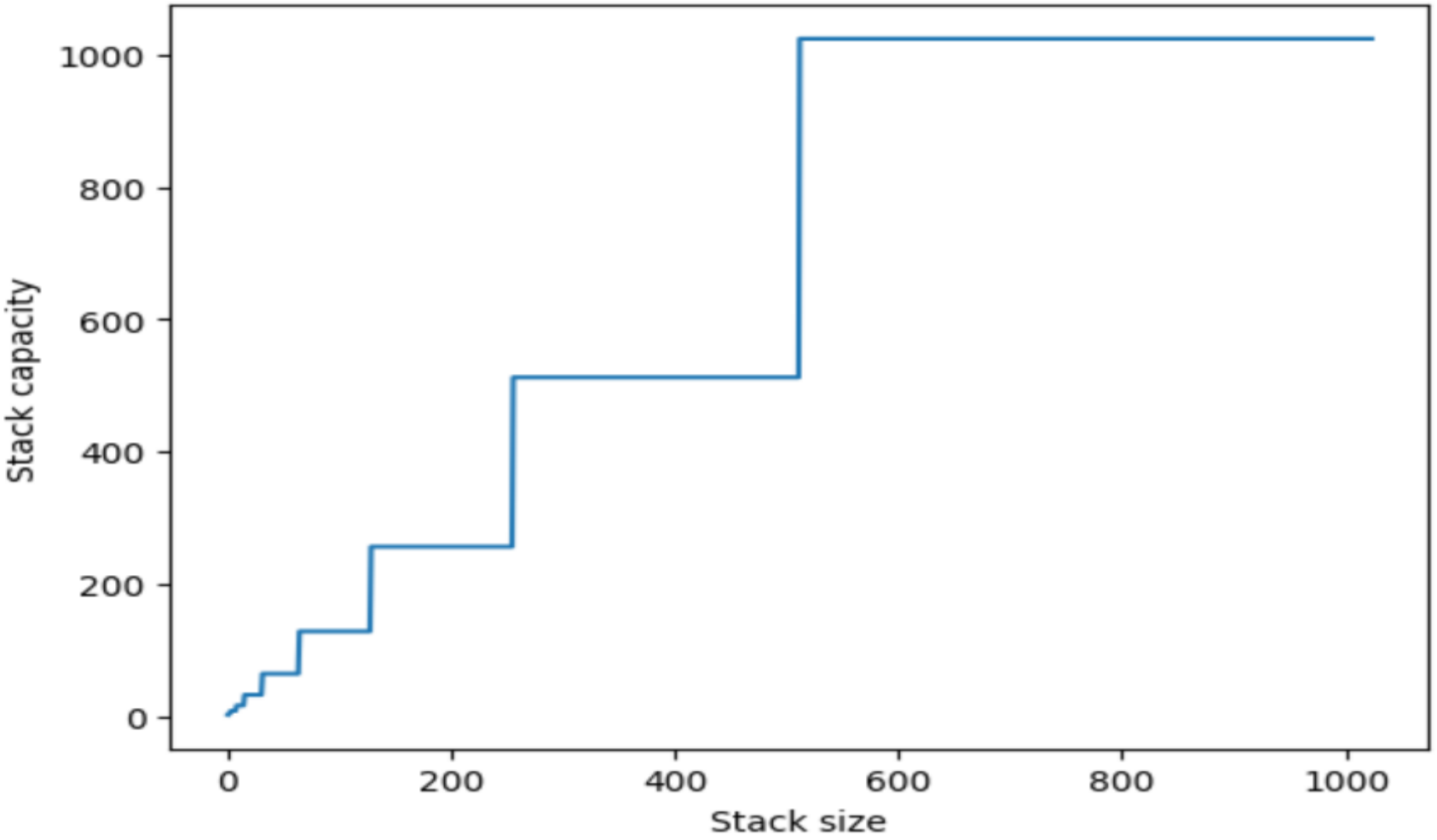
Analysis of Dynamic Stacks

- Assume get N (= 2^k) push() followed by N pop()
- T(2N) = Time for N push + Time for N pop
- = O(N) + O(N) = O(N)
- T(N) = O(N) – True or false?

Analysis of Dynamic Stacks

- Assume get $N = 2^k$ push() followed by N pop()
- T(2N) = Time for N push + Time for N pop
- = O(N) + O(N) = O(N)
- T(N) = O(N) – False.
- Because T(N) represents the time taken for the worst-case input

Analysis of Dynamic Stacks



Consider a sequence of 513 push() operations followed by 511 (pop() + push()) operation.

Dynamic Stacks: Worst case

- In general, consider 2^k+1 push() operations
- Followed by 2^{k-1} push()+pop() operations
- Total number of operations

  $2^k+1 + 2 * 2^{k-1} = 2^k+ 2^k +1 = 2^{k+1}+1 = N$
- Time taken by 2^k+1 push() operations

  $O(2^k+1) = O(N/2) = O(N)$
- Time taken by each push()+pop() operation

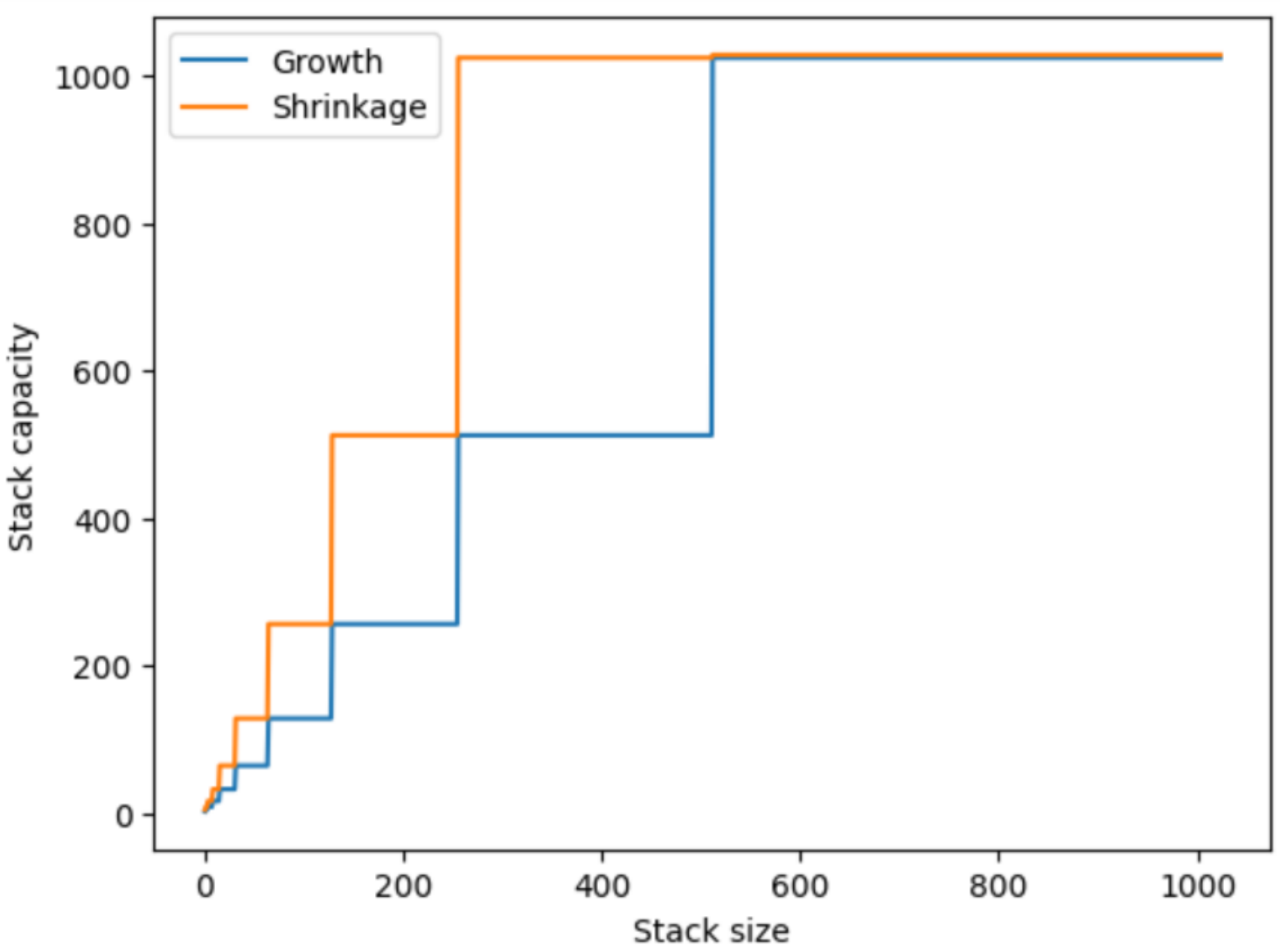
  $2^{k+1} + 2^k + 2^k = 2^{k+2} = 2N-2$
- Total time taken by 2^{k-1} [= (N-1)/4] push()+pop() operations

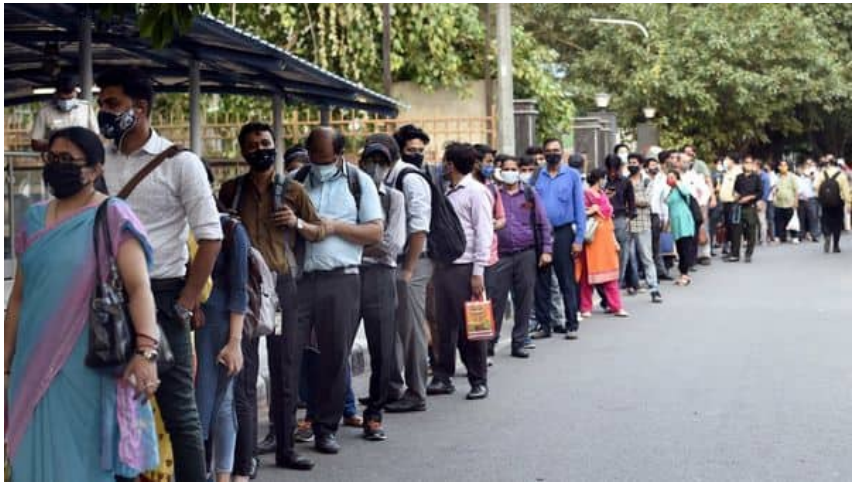
  $((N-1)/4) * (2N - 2) = O(N^2)$

Solution: Hysteresis

- Grow the stack as earlier
- While shrinking the stack, do not shrink when
  - size ≤ capacity / C
- Shrink when
  - size ≤ capacity / C^2

Dynamic Stack with Hysteresis





Queues and Lists

Images courtesy: www.123rf.com; www.livemint.com

Queues

- What is a Queue?
- Container of objects
  - Put an object (enqueue)
  - Take an object out (dequeue)
  - Order of objects: First-In First-Out (FIFO)
- Real life examples of a queue

Image credit: Jurgen Ziewe/Getty Images; Taken from: [Cue vs. Queue: How to Choose the Right Word (thoughtco.com)]

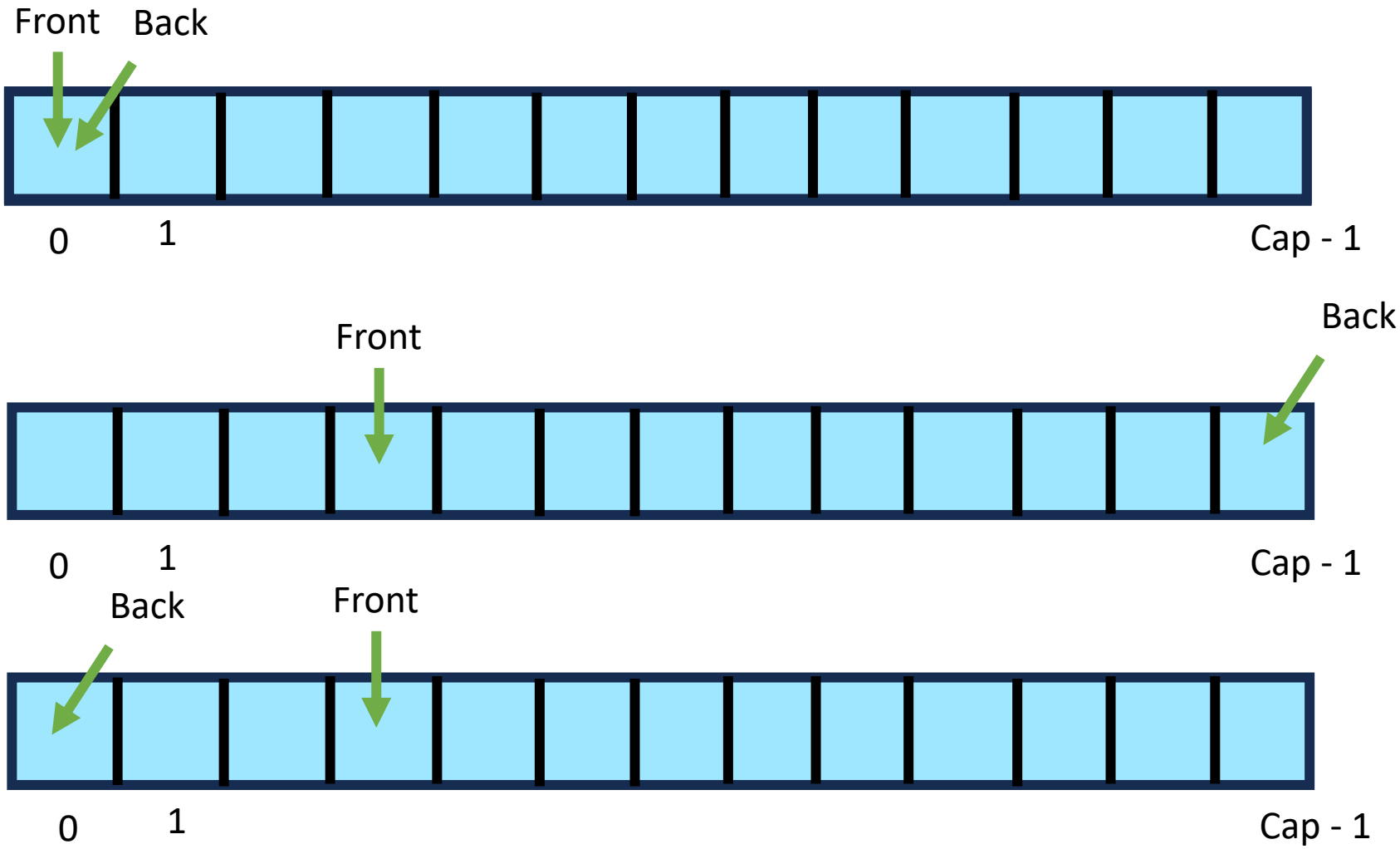
Female voters standing in a queue for casting their votes during the 3rd Phase of General Elections-2014, in New Delhi on April 10, 2014.
Image source: Wikipedia

Operations of Queues

- Enqueue
- Dequeue
- Front
- Size
- IsEmpty

Queue Implementation using Array

Queue Implementation using Array



Queue Implementation using Array

```
class Queue {
private:
    int f, b;
    int *Q;
    int cap;

    // First element is Q[f]
    // Q[b] is empty space where a new element is to be inserted
    // Last element is Q[(b - 1 + cap) mod cap
    // If f == b then Q is empty
    // If (b + 1) mod cap == f then Q is full
    // Else Q has space to insert elements
    // Number of elements Q can have: cap - 1
```

Queue Implementation using Array

```
public:

    Queue(int sz) {
        Q = new int[sz];  // TBD Check for errors
        cap = sz;
    }
    ~Queue() {
        delete Q;
    }
```

Queue Implementation using Array

```
void Enqueue(int element) {
    // First check for Q full
    if (((b + 1) % cap) == f) throw Q_FULL_EXCEPTION;
    else {
        Q[b] = element;
        b = (b + 1) % cap;
    }
}
```

Queue Implementation using Array

```
int Dequeue() {
    // First check for Q empty
    if (f == b) throw Q_EMPTY_EXCEPTION;
    else {
        int result;
        result = Q[f];
        f = (f + 1) % cap;
        return result;
    }
}
```

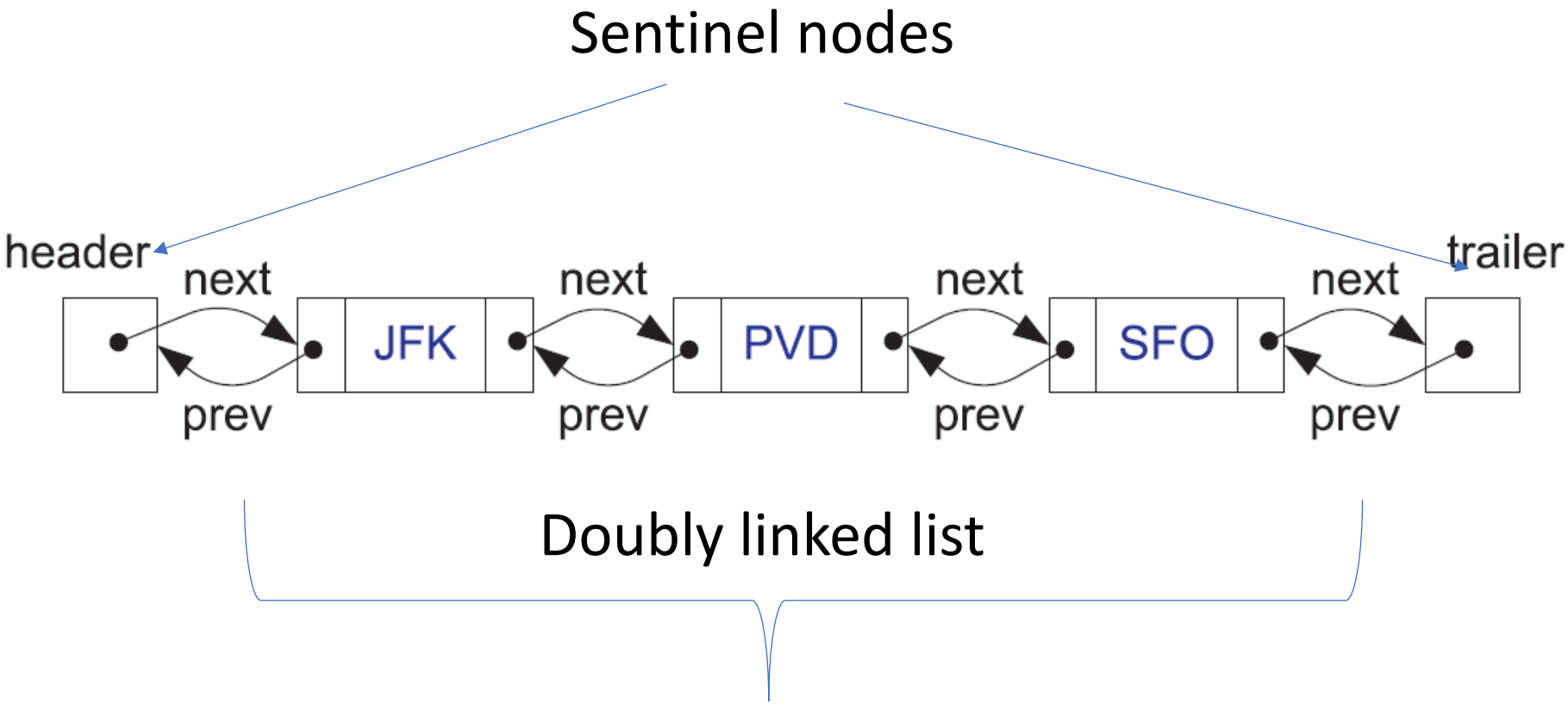
Queue Implementation using Array

```
int Front() {
    return Q[f];
}

int Size() {
    return ((f - b + cap) % cap);
}

bool isEmpty() {
    return (f == b);
}
```

Double Ended Queues (Deque)

- A queue that supports insertion and deletion at both the ends
- Operations on Deque
  - insertFront(e)
  - insertBack(e)
  - eraseFront(e)
  - eraseBack(e)
  - front()
  - back()
  - size()
  - empty()

Data Structure for Deque



Basic Operations Needed

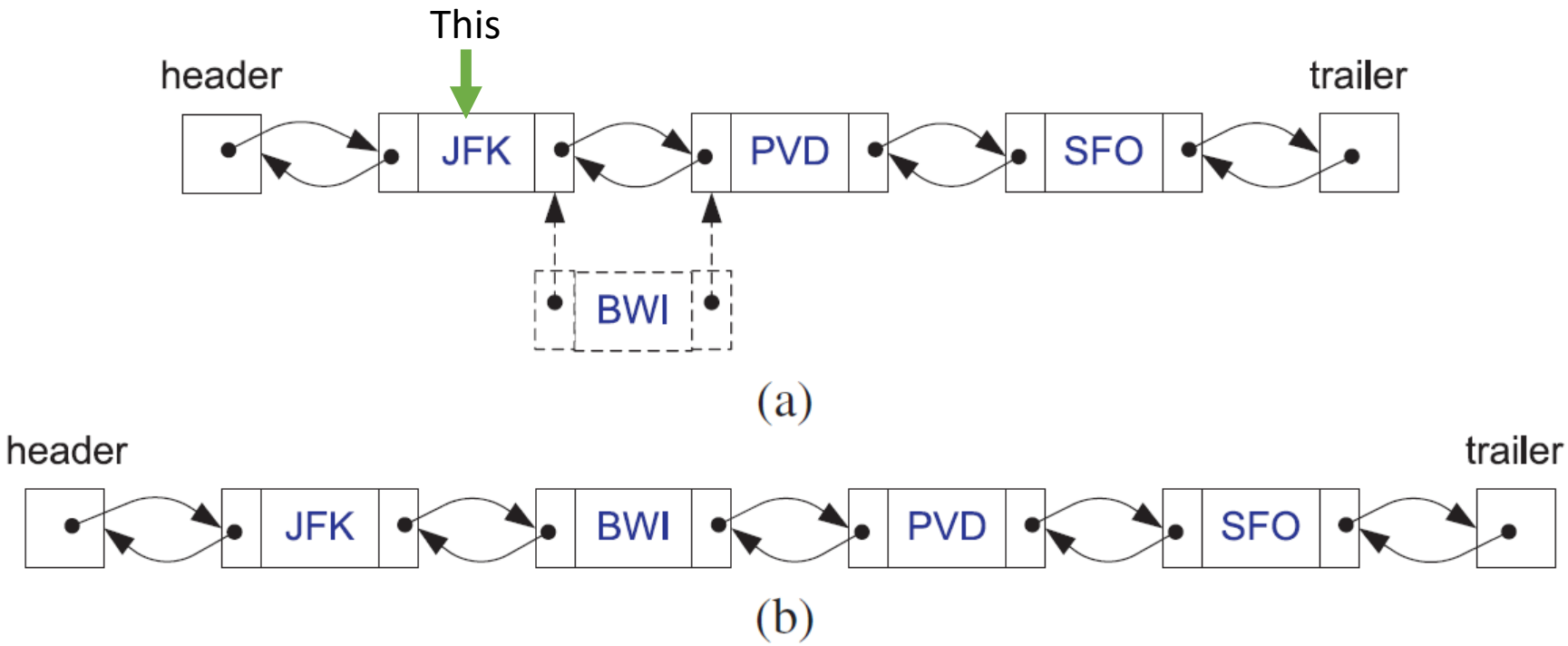
- insertBefore
- insertAfter
- DLLDelete

    - insertFront(): insertAfter(header)
    - insertBack(): insertBefore(trailer)
    - eraseFront(e): DLLDelete(header->next)
    - eraseBack(e): DLLDelete(trailer->prev)

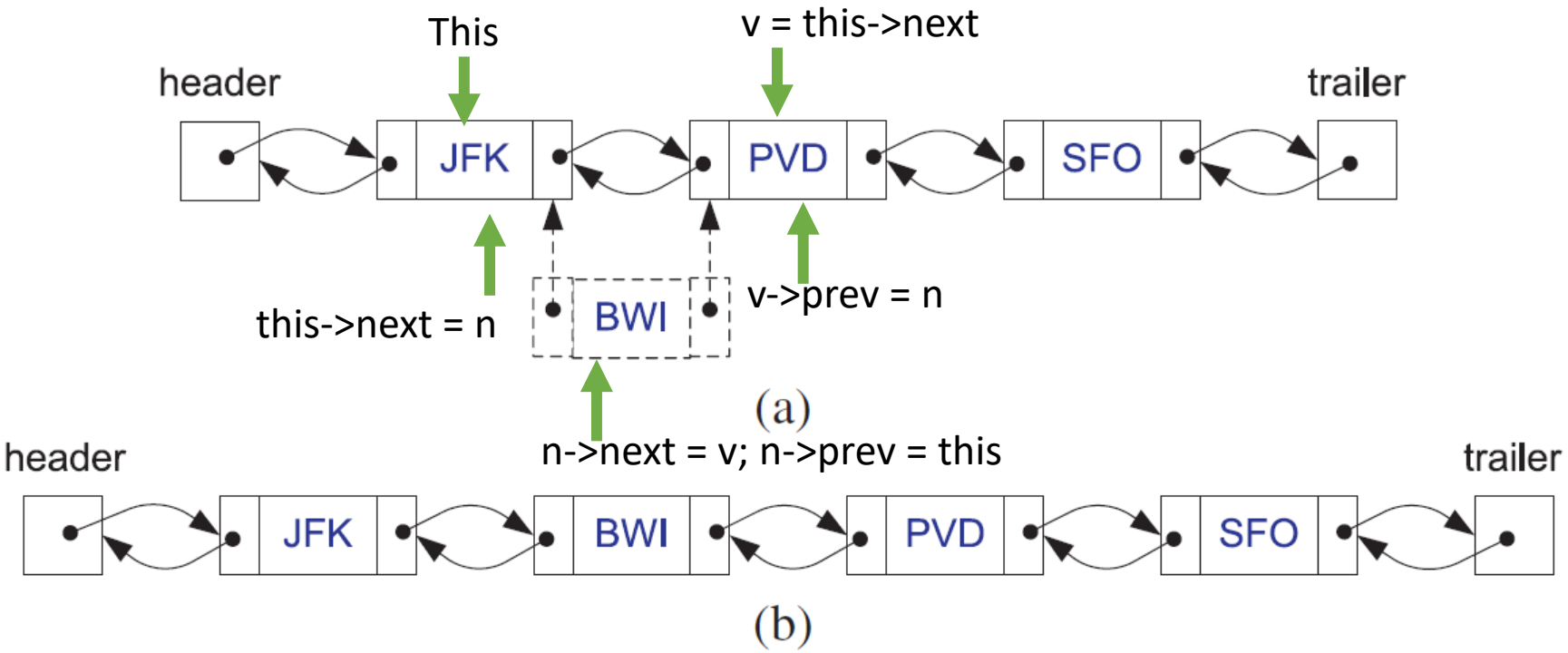
Deque using Doubly Linked Lists

```
class DLLNode {
    // The Doubly linked list will have two sentinal nodes header and trailer.
    // No data is kept in sentinal nodes
    // Sentinal node invariants: (i) header->prev == NULL (ii) trailer->next == NULL
    // The above invariants give a method to identify header and trailer
    // Non-sentinal node invariants:
    // (iii) n->next->prev == n (iv) n->prev->next == n

private:
    int data;

public:
    DLLNode *next, *prev;
    void insertAfter(DLLNode *n); // Insert node n after this  node
    void insertBefore(DLLNode *n); // Insert node n before this node
    void DLLdelete(); // Delete the this node
};
```

Insertion in a Doubly Linked List



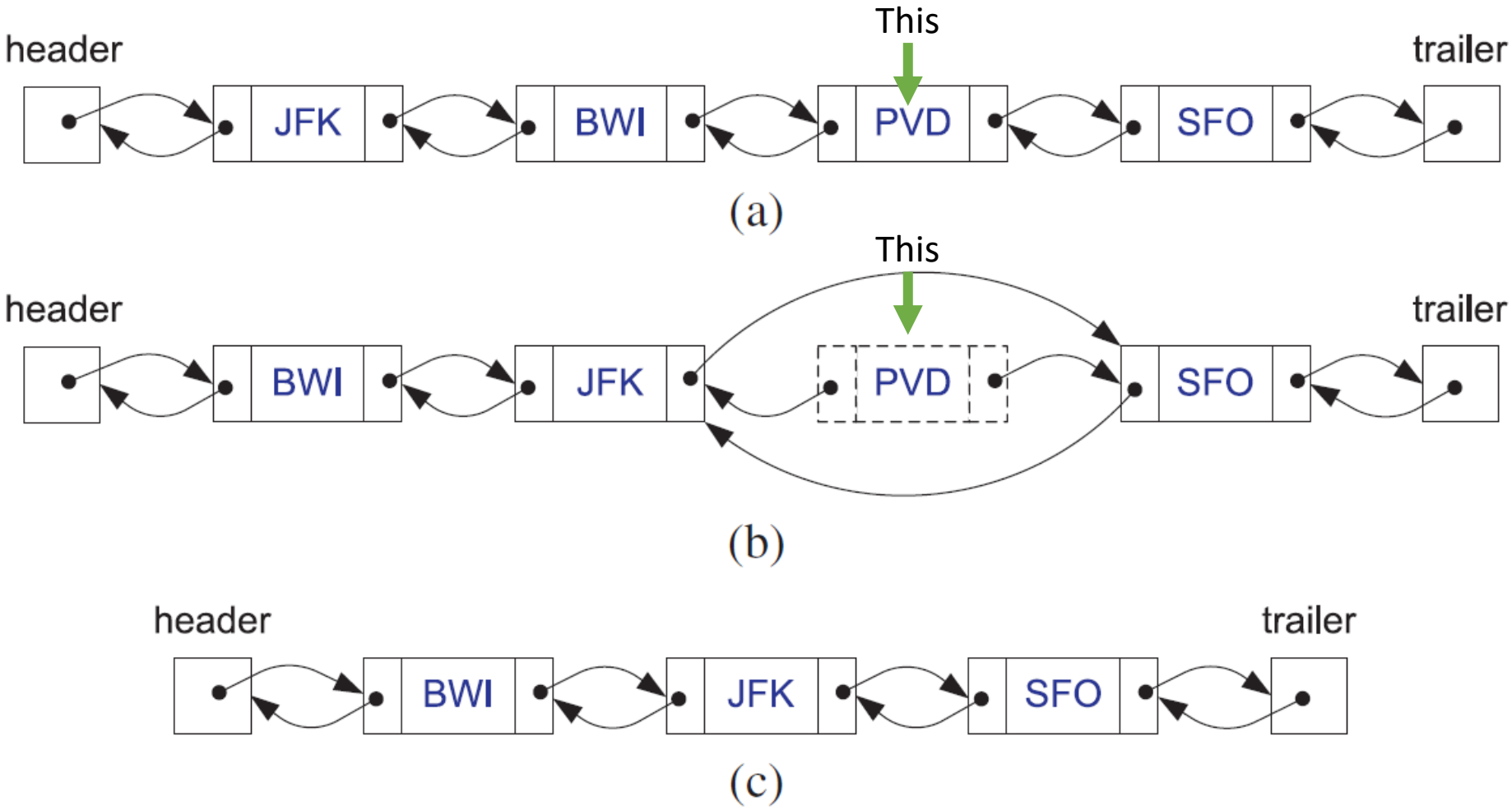
Insertion in a Doubly Linked List



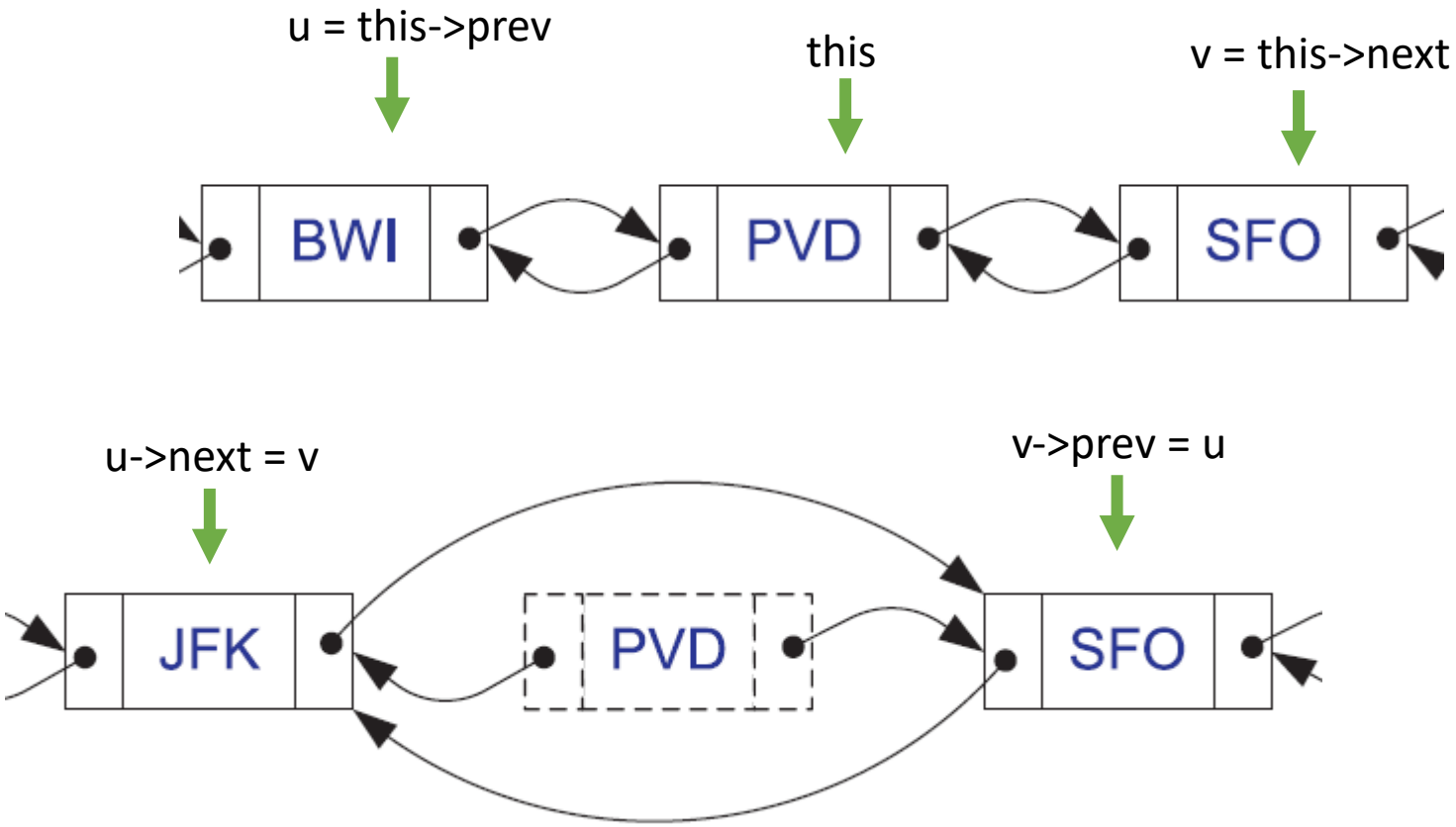
InsertAfter in a Doubly Linked List

```
void DLLNode::insertAfter(DLLNode *n) {

    n->next = next;
    n->prev = this;
    next = n;
    n->next->prev = n;

}
```

Deletion in a Doubly Linked List



DLLNode::Delete()



DLLNode::Delete()

```
void DLLNode::DLLdelete() {

    prev->next = next;
    next->prev = prev;

    delete this;

}
```

Stacks using Deque

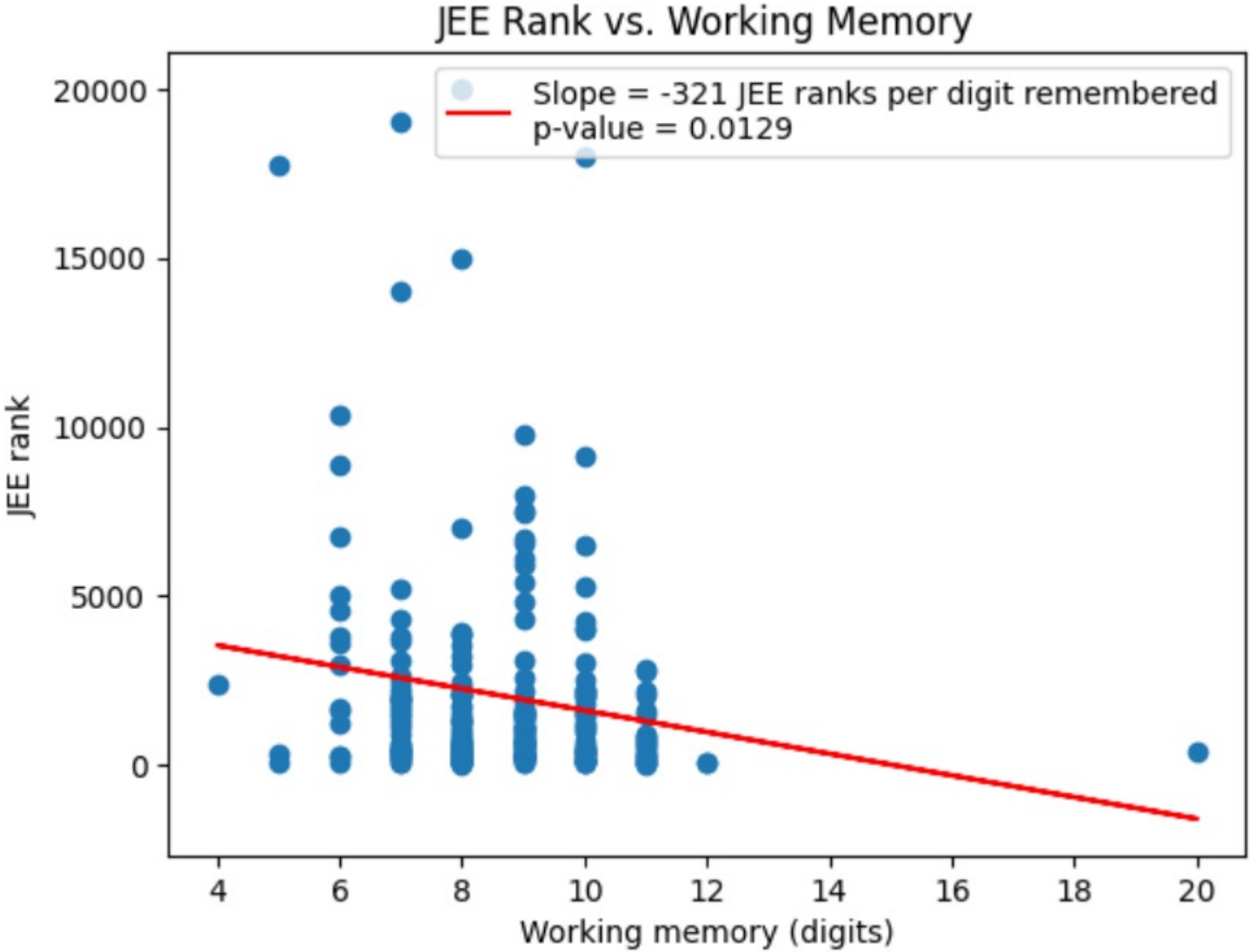
Stack Method	Deque Implementation
push(o)	insertFront(o)
pop()	eraseFront()
top()	front()
size()	size()
empty()	empty()

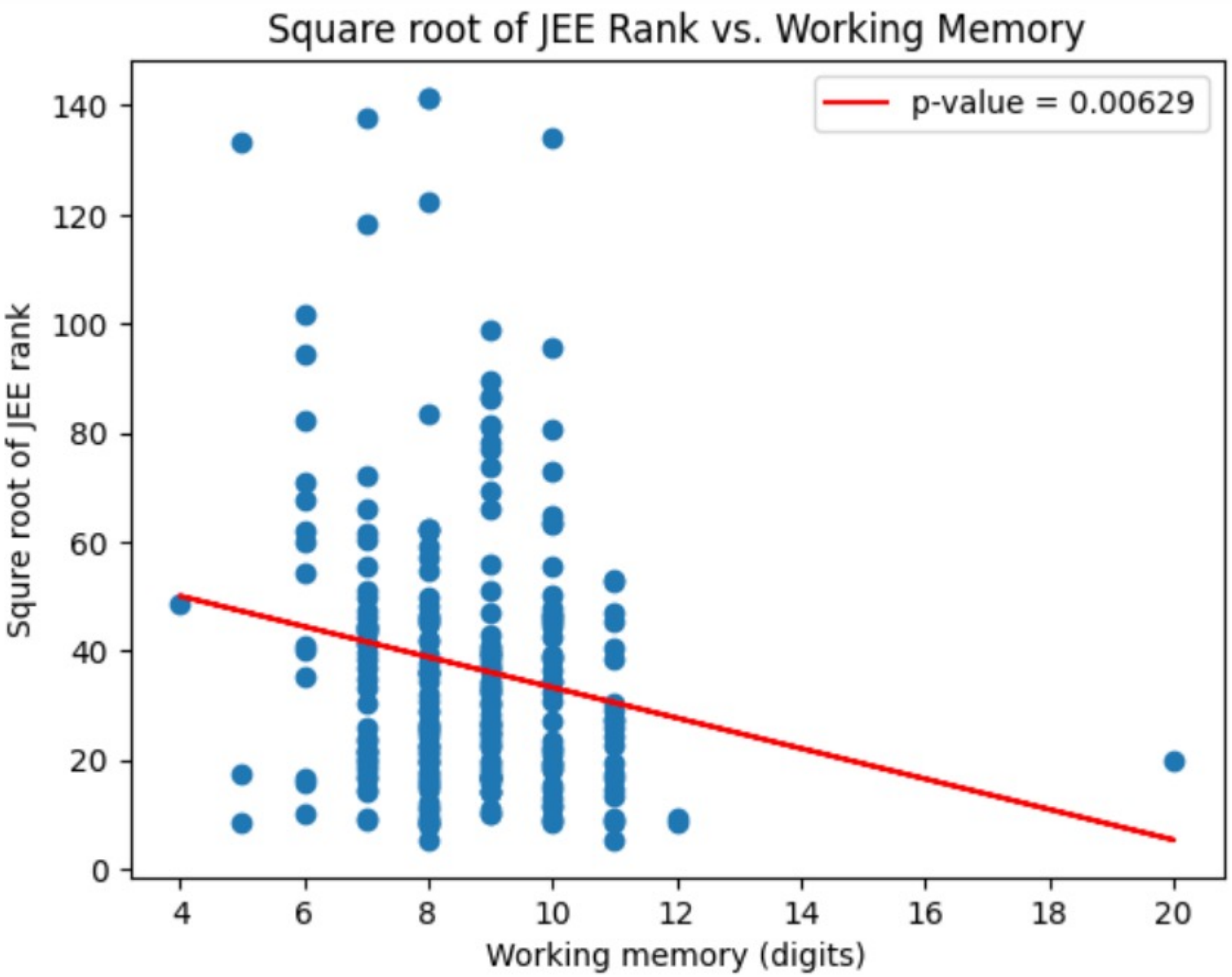
Queues using Deque

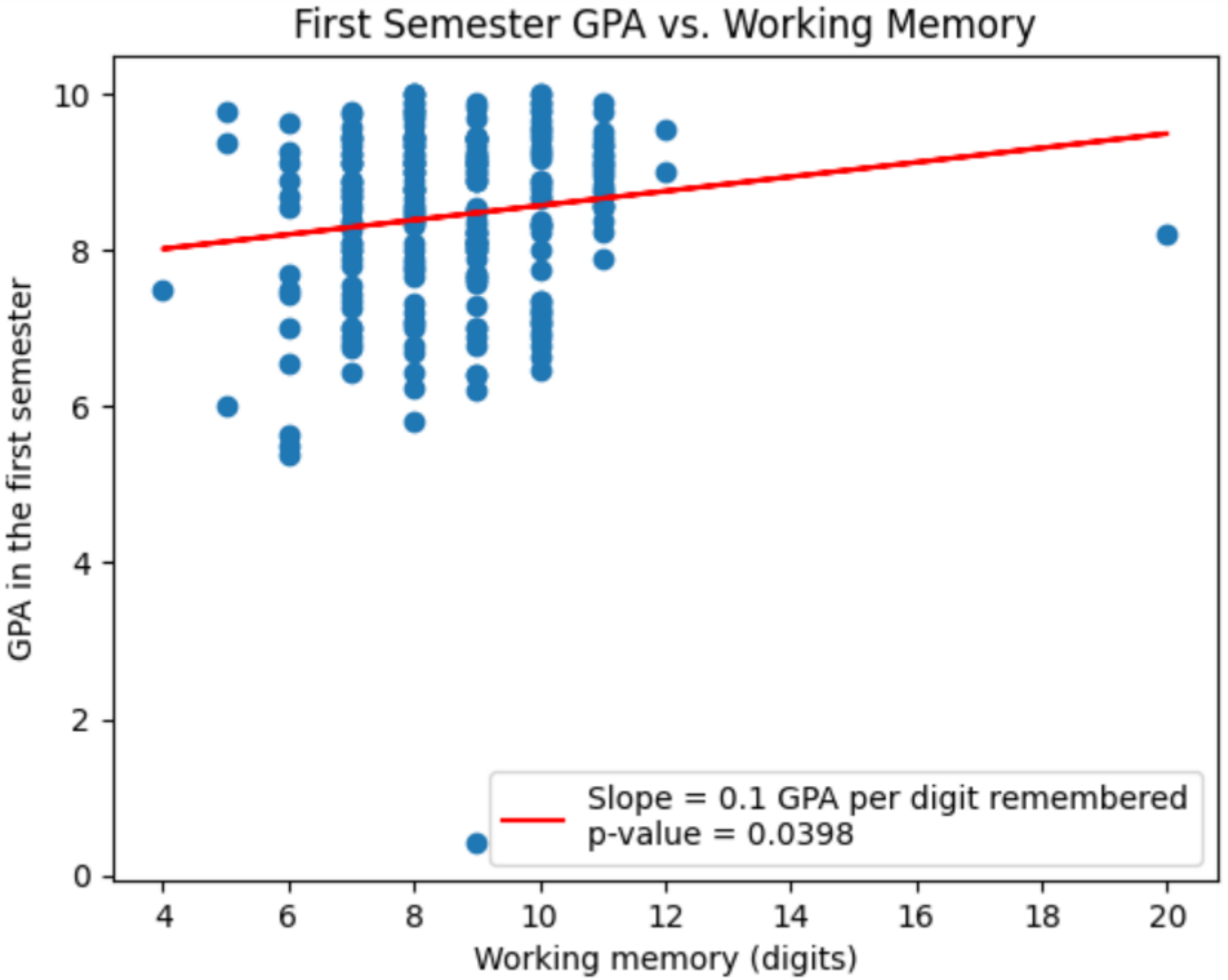
Queue Method	Deque Implementation
enqueue(o)	insertBack(o)
dequeue()	eraseFront()
front()	front()
size()	size()
empty()	empty()

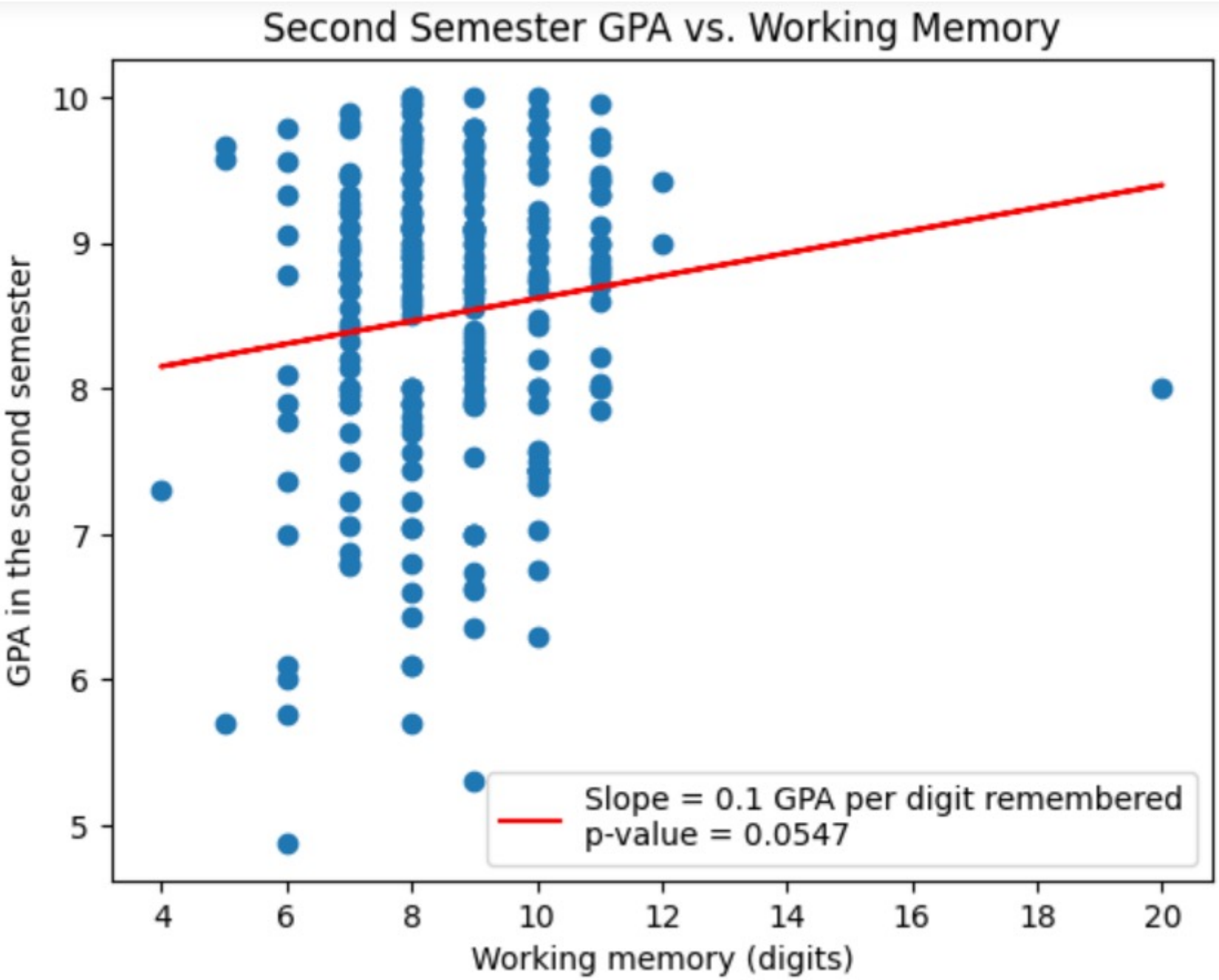
WM Experiment Results

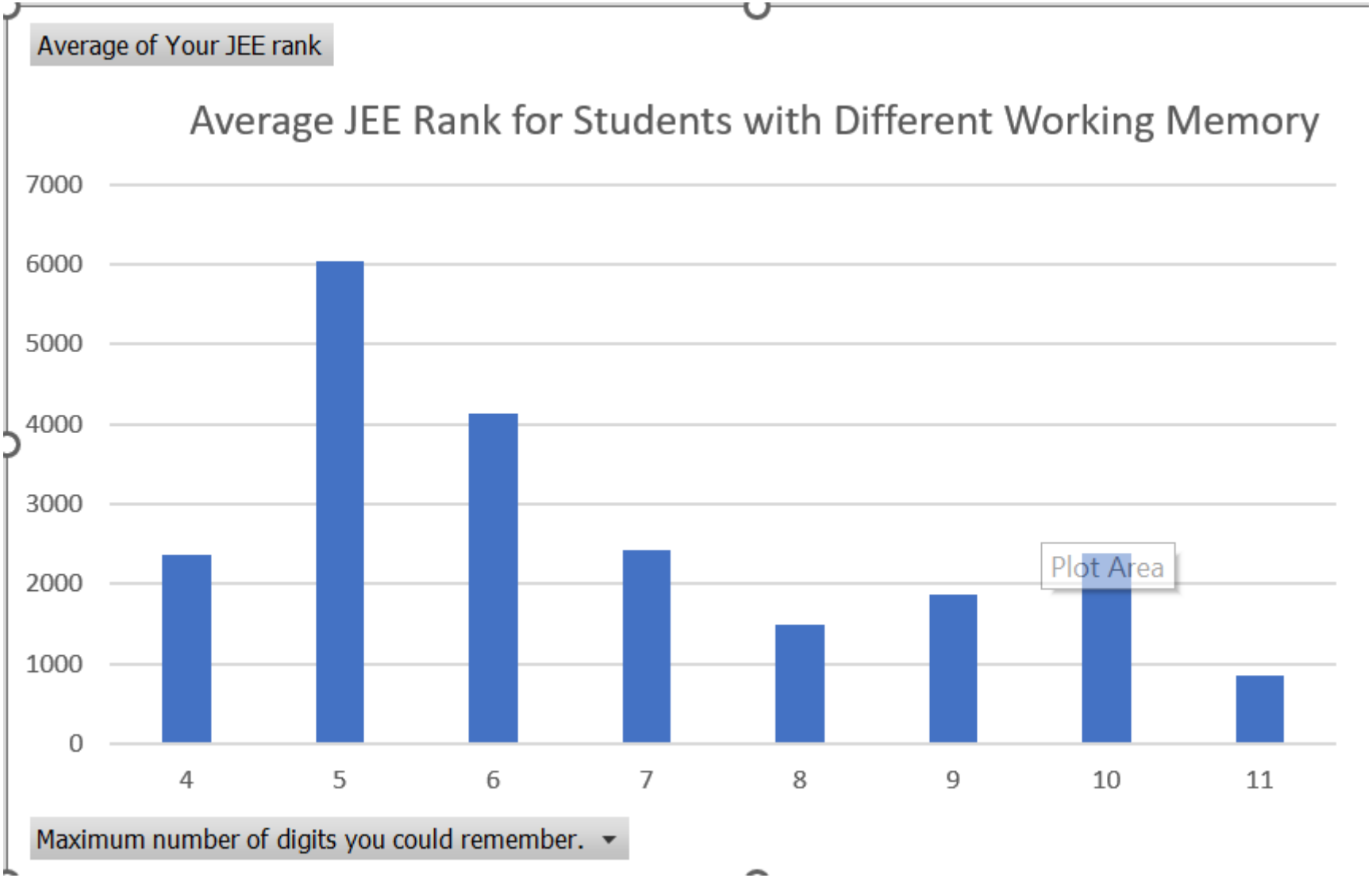
248 responses

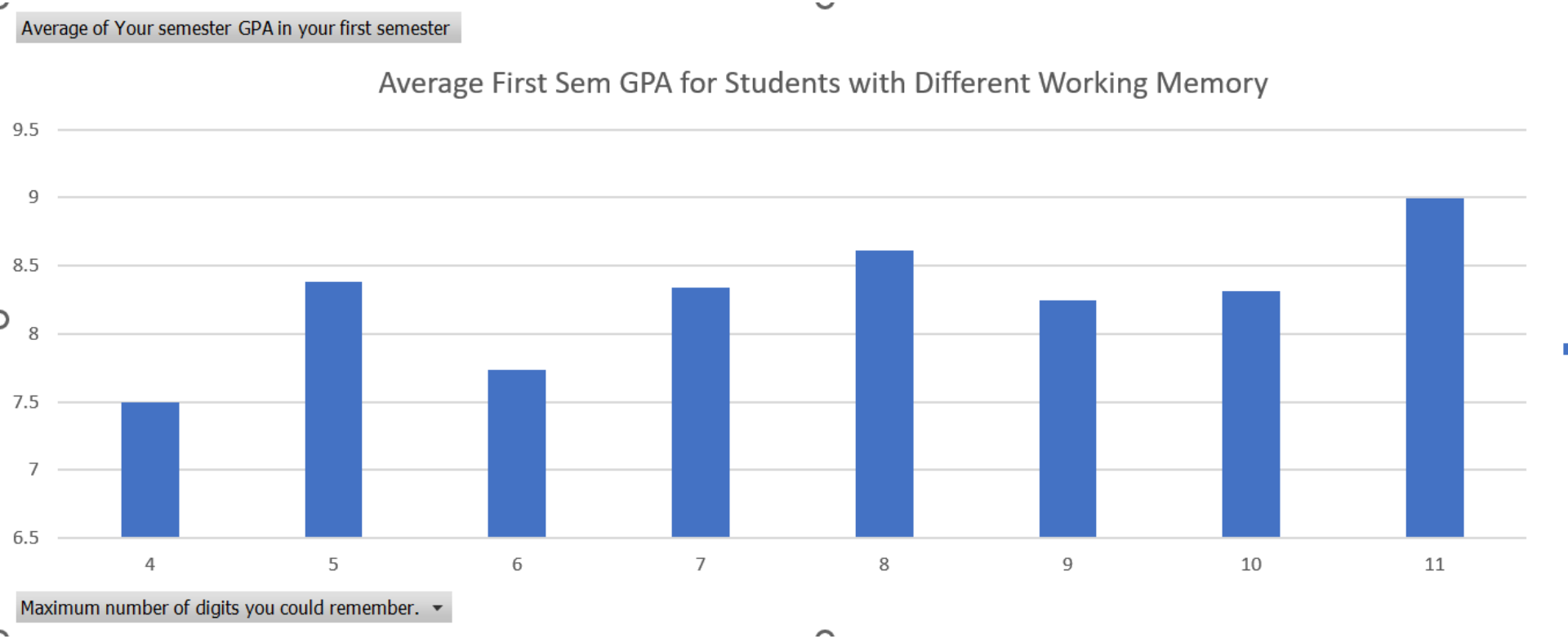

JEE Rank vs. Working Memory
Slope = -321 JEE ranks per digit remembered
p-value = 0.0129
20000
15000
10000
5000
0
JEE rank
4
6
8
10
12
14
16
18
20
Working memory (digits)

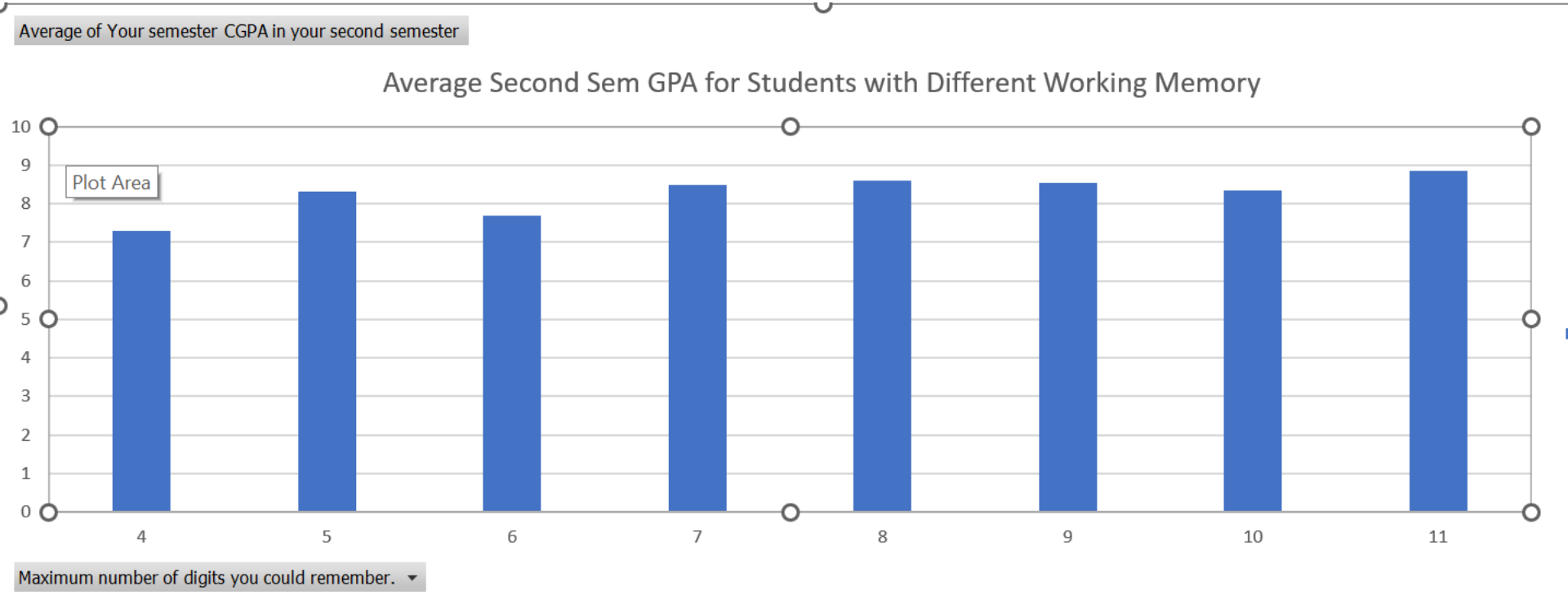

Square root of JEE Rank vs. Working Memory
p-value = 0.00629
Squre root of JEE rank
Working memory (digits)
0
20
40
60
80
100
120
140
4
6
8
10
12
14
16
18
20


First Semester GPA vs. Working Memory
GPA in the first semester
Working memory (digits)
0
2
4
6
8
10
4
6
8
10
12
14
16
18
20
Slope = 0.1 GPA per digit remembered
p-value = 0.0398


Second Semester GPA vs. Working Memory
GPA in the second semester
Working memory (digits)
Slope = 0.1 GPA per digit remembered
p-value = 0.0547
10
9
8
7
6
5
4
6
8
10
12
14
16
18
20


Average of Your JEE rank
Average JEE Rank for Students with Different Working Memory
7000
6000
5000
4000
3000
2000
1000
0
4
5
6
7
8
9
10
11
Plot Area
Maximum number of digits you could remember.


Average of Your semester GPA in your first semester
Average First Sem GPA for Students with Different Working Memory
9.5
9
8.5
8
7.5
7
6.5
4
5
6
7
8
9
10
11
Maximum number of digits you could remember.


Average of Your semester CGPA in your second semester
Average Second Sem GPA for Students with Different Working Memory
Plot Area
10
9
8
7
6
5
4
3
2
1
0
4
5
6
7
8
9
10
11
Maximum number of digits you could remember.

Ready for Quiz 1?

- Can only submit response once
- Cannot edit after submitting
- Responses submitted after five minutes of the start of quiz will get a zero mark
- No other queries will be entertained