****

University of Central Punjab

**Data Structure and Algorithms**

**Assignment 01**

**Name:** Abdullah Maqbool

**Reg no:** L1F22BSSE0391

**Section:** P4

**Submitted to:** Ma`am Afifa Hameed

# Scenario 01 (E-Commerce Website)

## Implementation Using ArrayList

import java.util.ArrayList;  
import java.util.Scanner;  
  
class Order {  
 int orderId;  
 double totalAmount;  
  
 public Order(int orderId, double totalAmount) {  
 this.orderId = orderId;  
 this.totalAmount = totalAmount;  
 }  
  
 public void printOrderDetails() {  
 System.*out*.println("Order Id: " + orderId);  
 System.*out*.println("Total Order Amount: " + totalAmount);  
 }  
}  
  
class OrderManagementSystem {  
 ArrayList<Order> ordersList;  
  
 public OrderManagementSystem() {  
 ordersList = new ArrayList<>();  
 }  
  
 public void addOrder(Order order) { // ~~> take O(n) time complexity because shifting required!  
 ordersList.addFirst(order); // inserting at first insert (implementing FIFO)  
 System.*out*.println("Order Added Successfully!");  
 }  
  
 public void processOrder() { // ~~> take O(1) time complexity!  
 if (ordersList.isEmpty()) {  
 System.*out*.println("Error: List is empty!");  
 } else {  
 ordersList.getLast().printOrderDetails();  
 ordersList.removeLast(); // removing from the last of the list (implementing FIFO)  
 System.*out*.println("Order Processed Successfully!");  
 }  
 }  
  
 public void printRecentOrderDetails() { // ~~> take O(1) time complexity  
 if (ordersList.isEmpty()) {  
 System.*out*.println("Error: List is empty!");  
 } else {  
 ordersList.getFirst().printOrderDetails();  
 }  
 }  
}  
  
  
public class Scenario1WithArrayList {  
 public static void main(String[] args) {  
 OrderManagementSystem OMS = new OrderManagementSystem();  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
 System.*out*.println(" \t ~~ Welcome to Order Management System(OMS) ~~ ");  
 while (true) {  
 System.*out*.println("\n 1) Store Customer Order");  
 System.*out*.println(" 2) Process Order");  
 System.*out*.println(" 3) Retrieve the most recent order details");  
 System.*out*.println(" 4) Exit");  
 System.*out*.println("\t Select an option (1,2,3 or 4)");  
 choice = sc.nextInt();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter Order id: ");  
 int orderId = sc.nextInt();  
 System.*out*.println("Enter total order amount: ");  
 double totalAmount = sc.nextDouble();  
 OMS.addOrder(new Order(orderId, totalAmount));  
 break;  
 case 2:  
 System.*out*.println("Details of the Order going to be processed: ");  
 OMS.processOrder();  
 break;  
 case 3:  
 System.*out*.println("Details of the most recent order: ");  
 OMS.printRecentOrderDetails();  
 break;  
 case 4:  
 System.*out*.println(" \t ~~ Thanks for using Order Management System(OMS)! ~~");  
 return;

default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Implementation Using LinkedList

import java.util.LinkedList;  
import java.util.Scanner;  
  
class Order {  
 int orderId;  
 double totalAmount;  
  
 public Order(int orderId, double totalAmount) {  
 this.orderId = orderId;  
 this.totalAmount = totalAmount;  
 }  
  
 public void printOrderDetails() {  
 System.*out*.println("Order Id: " + orderId);  
 System.*out*.println("Total Order Amount: " + totalAmount);  
 }  
}  
  
class OrderManagementSystem {  
 LinkedList<Order> ordersList;  
  
 public OrderManagementSystem() {  
 ordersList = new LinkedList<>();  
 }  
  
 public void addOrder(Order order) { // ~~> take O(1) time complexity because NO shifting required!  
 ordersList.addFirst(order); // inserting at first (implementing FIFO)  
 System.*out*.println("Order Added Successfully!");  
 }  
  
 public void processOrder() { // ~~> take O(1) time complexity!  
 if (ordersList.isEmpty()) {  
 System.*out*.println("Error: List is empty!");  
 } else {  
 ordersList.getLast().printOrderDetails();  
 ordersList.removeLast(); // removing from the last of the list (implementing FIFO)  
 System.*out*.println("Order Processed Successfully!");  
 }  
 }  
  
 public void printRecentOrderDetails() { // ~~> take O(1) time complexity  
 if (ordersList.isEmpty()) {  
 System.*out*.println("Error: List is empty!");  
 } else {  
 ordersList.getFirst().printOrderDetails();  
 }  
 }  
}  
  
public class Scenario1WithLinkedList {  
 public static void main(String[] args) {  
 OrderManagementSystem OMS = new OrderManagementSystem();  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
 System.*out*.println(" \t ~~ Welcome to Order Management System(OMS) ~~ ");  
 while (true) {  
 System.*out*.println("\n 1) Store Customer Order");  
 System.*out*.println(" 2) Process Order");  
 System.*out*.println(" 3) Retrieve the most recent order details");  
 System.*out*.println(" 4) Exit");  
 System.*out*.println("\t Select an option (1,2,3 or 4)");  
 choice = sc.nextInt();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter Order id: ");  
 int orderId = sc.nextInt();  
 System.*out*.println("Enter total order amount: ");  
 double totalAmount = sc.nextDouble();  
 OMS.addOrder(new Order(orderId, totalAmount));  
 break;  
 case 2:  
 System.*out*.println("Details of the Order going to be processed: ");  
 OMS.processOrder();  
 break;  
 case 3:  
 System.*out*.println("Details of the most recent order: ");  
 OMS.printRecentOrderDetails();  
 break;  
 case 4:  
 System.*out*.println(" \t ~~ Thanks for using Order Management System(OMS)! ~~");  
 return;

default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Time Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Operation** | **ArrayList** | **LinkedList** |
| AddFirst(Order) | **O(n)** because shifting required | **O(1)** as NO shifting required |
| RemoveLast() | O(1) | O(1) |
| getMostRecentOrder() | O(1) | O(1) |

## Which implementation is better?

**LinkedList is better** for FIFO order processing because it takes **O(1)** time complexity for addition at first.

# Scenario 02 (Stock Market Application)

## Implementation Using ArrayList

import java.util.ArrayList;  
import java.util.Scanner;  
  
class StockManagementSystem {  
 ArrayList<Double> stockPricesList;  
  
 public StockManagementSystem() {  
 stockPricesList = new ArrayList<>();  
 }  
  
 public void addPrice(double stockPrice) { // time complexity - O(1) amortised  
 stockPricesList.add(stockPrice);  
 System.*out*.println("Stock price added in the list Successfully!");  
 }  
  
 public void removeOutdatedPrice() { // time complexity - O(n) as shifting required!  
 if (stockPricesList.isEmpty()) {  
 System.*out*.println("Error: No stock prices in the list!");  
 } else {  
 stockPricesList.removeFirst();  
 System.*out*.println("Outdated price removed Successfully!");  
 }  
 }  
  
 public double getHighestPrice() { // time complexity - O(n) as traversing required  
 if (stockPricesList.isEmpty()) {  
 return -1;  
 } else {  
 double temp = 0;  
 for (double i : stockPricesList) {  
 if (i > temp) {  
 temp = i;  
 }  
 }  
 return temp;  
 }  
 }  
  
 public double getMinimumPrice() { // time complexity - O(n) as traversing required  
 if (stockPricesList.isEmpty()) {  
 return -1;  
 } else {  
 double temp = stockPricesList.getFirst();  
 for (double i : stockPricesList) {  
 if (i < temp) {  
 temp = i;  
 }  
 }  
 return temp;  
 }  
 }  
}  
  
public class Scenario2WithArrayList {  
 public static void main(String[] args) {  
 StockManagementSystem SMS = new StockManagementSystem();  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
  
 System.*out*.println("\t ~~ Welcome to Stock Market Application ~~");  
 while (true) {  
 System.*out*.println("\n 1) Add stock price");  
 System.*out*.println(" 2) Remove Outdated stock price");  
 System.*out*.println(" 3) Get Maximum stock price");  
 System.*out*.println(" 4) Get Minimum stock price");  
 System.*out*.println(" 5) Exit");  
 System.*out*.print("Select an option (1,2,3,4 or 5): ");  
 choice = sc.nextInt();  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter stock price: ");  
 double price = sc.nextDouble();  
 SMS.addPrice(price);  
 break;  
 case 2:  
 SMS.removeOutdatedPrice();  
 break;  
 case 3:  
 if (SMS.getHighestPrice() == -1) {  
 System.*out*.println("Error: No stock prices in the list!");  
 } else {  
 System.*out*.println("Highest stock price: " + SMS.getHighestPrice());  
 }  
 break;  
 case 4:  
 if (SMS.getMinimumPrice() == -1) {  
 System.*out*.println("Error: No stock prices in the list!");  
 } else {  
 System.*out*.println("Minimum stock price: " + SMS.getMinimumPrice());  
 }  
 break;  
 case 5:  
 System.*out*.println("\t ~~ Thanks for using Stock Market Application(SMS)");  
 return;

default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Implementation Using LinkedList

import java.util.LinkedList;  
import java.util.Scanner;  
  
class StockManagementSystem {  
 LinkedList<Double> stockPricesList;  
  
 public StockManagementSystem() {  
 stockPricesList = new LinkedList<>();  
 }  
  
 public void addPrice(double stockPrice) { // time complexity - O(1) amortised  
 stockPricesList.add(stockPrice);  
 System.*out*.println("Stock price added in the list Successfully!");  
 }  
  
 public void removeOutdatedPrice() { // time complexity - O(1) as NO shifting required!  
 if (stockPricesList.isEmpty()) {  
 System.*out*.println("Error: No stock prices in the list!");  
 } else {  
 stockPricesList.removeFirst();  
 System.*out*.println("Outdated price removed Successfully!");  
 }  
 }  
  
 public double getHighestPrice() { // time complexity - O(n) as traversing required  
 if (stockPricesList.isEmpty()) {  
 return -1;  
 } else {  
 double temp = 0;  
 for (double i : stockPricesList) {  
 if (i > temp) {  
 temp = i;  
 }  
 }  
 return temp;  
 }  
 }  
  
 public double getMinimumPrice() { // time complexity - O(n) as traversing required  
 if (stockPricesList.isEmpty()) {  
 return -1;  
 } else {  
 double temp = stockPricesList.getFirst();  
 for (double i : stockPricesList) {  
 if (i < temp) {  
 temp = i;  
 }  
 }  
 return temp;  
 }  
 }  
}  
  
public class Scenario2WithLinkedList {  
 public static void main(String[] args) {  
 StockManagementSystem SMS = new StockManagementSystem();  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
  
 System.*out*.println("\t ~~ Welcome to Stock Market Application ~~");  
 while (true) {  
 System.*out*.println("\n 1) Add stock price");  
 System.*out*.println(" 2) Remove Outdated stock price");  
 System.*out*.println(" 3) Get Maximum stock price");  
 System.*out*.println(" 4) Get Minimum stock price");  
 System.*out*.println(" 5) Exit");  
 System.*out*.print("Select an option (1,2,3,4 or 5): ");  
 choice = sc.nextInt();  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter stock price: ");  
 double price = sc.nextDouble();  
 SMS.addPrice(price);  
 break;  
 case 2:  
 SMS.removeOutdatedPrice();  
 break;  
 case 3:  
 if (SMS.getHighestPrice() == -1) {  
 System.*out*.println("Error: No stock prices in the list!");  
 } else {  
 System.*out*.println("Highest stock price: " + SMS.getHighestPrice());  
 }  
 break;  
 case 4:  
 if (SMS.getMinimumPrice() == -1) {  
 System.*out*.println("Error: No stock prices in the list!");  
 } else {  
 System.*out*.println("Minimum stock price: " + SMS.getMinimumPrice());  
 }  
 break;  
 case 5:  
 System.*out*.println("\t ~~ Thanks for using Stock Market Application(SMS)");  
 return;

default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Time Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Operation** | **ArrayList** | **LinkedList** |
| addPrice(price) | O(1) amortized | O(1) |
| removeFirst() | O(n) as shifting required | O(1) |
| Get minimum/maximum | O(n) | O(n) |

## Would a **Sorted Data Structure** improve performance?

**Definitely Yes**, because we can access the minimum price by just “list.get(0)” that takes O(1) time complexity, and the maximum by just “list.get(size – 1)” that also takes O(1) time complexity.

# Scenario 03 (Train Booking System)

## Implementation Using Array

import java.util.Scanner;  
  
class SeatBookingSystem {  
 String[] seats;  
 int totalSeats;  
  
 public SeatBookingSystem(int totalSeats) {  
 this.totalSeats = totalSeats;  
 seats = new String[totalSeats];  
 }  
  
 public void bookSeat(int seatNumber, String passengerName) { // time complexity - O(1)  
 if (seatNumber >= 0 && seatNumber < totalSeats) {  
 if(seats[seatNumber]==null) {  
 seats[seatNumber] = passengerName;  
 System.*out*.println("Seat number " + seatNumber + " booked successfully!");  
 }  
 else{  
 System.*out*.println("Sorry, Seat is already booked!");  
 }  
 } else {  
 System.*out*.println("Error: Invalid seat number.");  
 }  
 }  
  
 public void cancelSeat(int seatNumber) { // time complexity - O(1)  
 if (seatNumber >= 0 && seatNumber < totalSeats) {  
 if(seats[seatNumber]!=null){  
 seats[seatNumber] = null;  
 System.*out*.println("Seat number " + seatNumber + " Canceled successfully!");  
 }else{  
 System.*out*.println("This seat is never booked before!");  
 }  
 } else {  
 System.*out*.println("Error: Invalid seat number.");  
 }  
 }  
  
 public void searchAvailableSeats() { // time complexity - O(n)  
 System.*out*.print("Available seats are (seat number): ");  
 boolean allSeatsBooked = true;  
 for (int i = 0; i < totalSeats; i++) {  
 if (seats[i] == null) {  
 System.*out*.print(i + ",");  
 allSeatsBooked = false;  
 }  
 }  
 if (allSeatsBooked)  
 System.*out*.println("No seats available!");  
 }  
}  
  
public class Scenario3WithArray {  
 public static void main(String[] args) {  
 SeatBookingSystem SBS = new SeatBookingSystem(10);  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
 System.*out*.println("\t ~~ Welcome to Train seat booking system ~~");  
 while(true){  
 System.*out*.println("\n 1) Book a seat");  
 System.*out*.println(" 2) Cancel a booking");  
 System.*out*.println(" 3) View available seats");  
 System.*out*.println(" 4) Exit");  
 System.*out*.print("Select an option(1,2,3 or 4): ");  
 choice = sc.nextInt();  
 switch (choice){  
 case 1:  
 System.*out*.print("Enter Passenger Name: ");  
 String passengerName = sc.nextLine().trim();  
 sc.nextLine();  
 System.*out*.print("Enter a seat number (between 0 and 9): ");  
 int seatNumber = sc.nextInt();  
 SBS.bookSeat(seatNumber,passengerName);  
 break;  
 case 2:  
 System.*out*.print("Enter seat number to cancel booking: ");  
 int cancelSeatNumber = sc.nextInt();  
 SBS.cancelSeat(cancelSeatNumber);  
 break;  
 case 3:  
 SBS.searchAvailableSeats();  
 break;  
 case 4:  
 System.*out*.println("\t ~~ Thanks for using train seat booking system ~~");  
 return;

default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Implementation Using ArrayList

import java.util.ArrayList;  
import java.util.Scanner;  
  
class SeatBookingSystem {  
 ArrayList<String> seats;  
  
 public SeatBookingSystem(int totalSeats) {  
 this.seats = new ArrayList<>(totalSeats);  
 for (int i = 0; i < totalSeats; i++) {  
 seats.add(null);  
 }  
 }  
  
 public void bookSeat(int seatNumber, String passengerName) { // time complexity - O(1)  
 if (seatNumber >= 0 && seatNumber < seats.size()) {  
 if (seats.get(seatNumber) == null) {  
 seats.set(seatNumber, passengerName);  
 System.*out*.println("Seat number " + seatNumber + " booked successfully!");  
 } else {  
 System.*out*.println("Sorry, Seat is already booked!");  
 }  
 } else {  
 System.*out*.println("Error: Invalid seat number.");  
 }  
 }  
  
 public void cancelSeat(int seatNumber) { // time complexity - O(1)  
 if (seatNumber >= 0 && seatNumber < seats.size()) {  
 if (seats.get(seatNumber) != null) {  
 seats.set(seatNumber, null);  
 System.*out*.println("Seat number " + seatNumber + " canceled successfully!");  
 } else {  
 System.*out*.println("This seat was never booked before!");  
 }  
 } else {  
 System.*out*.println("Error: Invalid seat number.");  
 }  
 }  
  
 public void searchAvailableSeats() { // time complexity - O(n)  
 System.*out*.print("Available seats are (seat number): ");  
 boolean allSeatsBooked = true;  
 for (int i = 0; i < seats.size(); i++) {  
 if (seats.get(i) == null) {  
 System.*out*.print(i + ", ");  
 allSeatsBooked = false;  
 }  
 }  
 if (allSeatsBooked) {  
 System.*out*.println("No seats available!");  
 }  
 }  
}  
  
public class Scenario3WithArrayList {  
 public static void main(String[] args) {  
 SeatBookingSystem SBS = new SeatBookingSystem(10);  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
  
 System.*out*.println("\t ~~ Welcome to Train Seat Booking System ~~");  
 while (true) {  
 System.*out*.println("\n 1) Book a seat");  
 System.*out*.println(" 2) Cancel a booking");  
 System.*out*.println(" 3) View available seats");  
 System.*out*.println(" 4) Exit");  
 System.*out*.print("Select an option (1, 2, 3, or 4): ");  
 choice = sc.nextInt();  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter Passenger Name: ");  
 String passengerName = sc.nextLine().trim();  
 System.*out*.print("Enter a seat number (between 0 and 9): ");  
 int seatNumber = sc.nextInt();  
 sc.nextLine();  
 SBS.bookSeat(seatNumber, passengerName);  
 break;  
 case 2:  
 System.*out*.print("Enter seat number to cancel booking: ");  
 int cancelSeatNumber = sc.nextInt();  
 sc.nextLine();  
 break;  
 case 3:  
 SBS.searchAvailableSeats();  
 break;  
 case 4:  
 System.*out*.println("\t ~~ Thanks for using train seat booking system ~~");  
 sc.close();  
 return;  
 default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Time Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Operation** | **ArrayList** | **LinkedList** |
| bookSeat(seatNumber) | O(1) | O(1) |
| cancelSeat(seatNumber) | O(1) | O(1) |
| searchAvailableSeats() | O(n) | O(n) |

## Which approach is better for a large-scale system?

ArrayList-based approach is better for large-scale train booking systems because:

1. Scalability: Dynamically grows to accommodate more seats or trains.

2. Flexibility: Supports future enhancements with minimal changes.

# Scenario 04 (Social Medial App comment section)

## Implementation Using LinkedList

import java.util.LinkedList;  
import java.util.Scanner;  
  
class CommentSection {  
 LinkedList<String> comments;  
  
 public CommentSection() {  
 this.comments = new LinkedList<>();  
 }  
  
 public void addComment(String comment) { // time complexity - O(1)  
 comments.addLast(comment); // Adds to the end (chronological order)  
 System.*out*.println("Comment added: " + comment);  
 }  
  
 public void editComment(int index, String newComment) { // time complexity - O(n)  
 if (index >= 0 && index < comments.size()) {  
 comments.set(index, newComment);  
 System.*out*.println("Comment at index " + index + " updated to: " + newComment);  
 } else {  
 System.*out*.println("Error: Invalid comment index.");  
 }  
 }  
  
 public void deleteComment(int index) { // time complexity - O(n)  
 if (index >= 0 && index < comments.size()) {  
 String removed = comments.remove(index);  
 System.*out*.println("Comment deleted: " + removed);  
 } else {  
 System.*out*.println("Error: Invalid comment index.");  
 }  
 }  
  
 public void searchComment(String keyword) { // time complexity - O(n)  
 System.*out*.println("Comments containing '" + keyword + "':");  
 boolean found = false;  
 for (int i = 0; i < comments.size(); i++) {  
 if (comments.get(i).toLowerCase().contains(keyword.toLowerCase())) {  
 System.*out*.println("Index " + i + ": " + comments.get(i));  
 found = true;  
 }  
 }  
 if (!found) {  
 System.*out*.println("No comments found with keyword '" + keyword + "'.");  
 }  
 }  
  
 public void displayReverseOrder() { // time complexity - O(n)  
 System.*out*.println("Comments (most recent first):");  
 for (int i = comments.size() - 1; i >= 0; i--) {  
 System.*out*.println("Index " + i + ": " + comments.get(i));  
 }  
 }  
  
 public void displayChronologicalOrder() { // time complexity - O(n)  
 System.*out*.println("Comments (chronological order):");  
 for (int i = 0; i < comments.size(); i++) {  
 System.*out*.println("Index " + i + ": " + comments.get(i));  
 }  
 }  
}  
  
public class Scenario4WithLinkedList {  
 public static void main(String[] args) {  
 CommentSection cs = new CommentSection();  
 Scanner sc = new Scanner(System.*in*);  
 int choice;  
  
 System.*out*.println("\t~~ Welcome to the Comment Section ~~");  
 while (true) {  
 System.*out*.println("\n1) Add a comment");  
 System.*out*.println("2) Edit a comment");  
 System.*out*.println("3) Delete a comment");  
 System.*out*.println("4) Search for a comment by keyword");  
 System.*out*.println("5) Display comments (most recent first)");  
 System.*out*.println("6) Exit");  
 System.*out*.print("Select an option (1-6): ");  
 choice = sc.nextInt();  
 sc.nextLine();  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter your comment: ");  
 String comment = sc.nextLine();  
 cs.addComment(comment);  
 break;  
 case 2:  
 System.*out*.print("Enter the index of the comment to edit: ");  
 int editIndex = sc.nextInt();  
 sc.nextLine();   
 System.*out*.print("Enter the new comment: ");  
 String newComment = sc.nextLine();  
 cs.editComment(editIndex, newComment);  
 break;  
 case 3:  
 System.*out*.print("Enter the index of the comment to delete: ");  
 int deleteIndex = sc.nextInt();  
 cs.deleteComment(deleteIndex);  
 break;  
 case 4:  
 System.*out*.print("Enter a keyword to search: ");  
 String keyword = sc.nextLine();  
 cs.searchComment(keyword);  
 break;  
 case 5:  
 cs.displayReverseOrder();  
 break;  
 case 6:  
 System.*out*.println("\t~~ Thanks for using the Comment Section ~~");  
 sc.close();  
 return;  
 default:  
 System.*out*.println("Invalid option. Please try again.");  
 }  
 }  
 }  
}

## Time Complexity Analysis

|  |  |
| --- | --- |
| **Operation** | **ArrayList** |
| Insertion | O(1) |
| Deletion | O(n) |
| Searching | O(n) |
| Editing | O(n) |
| Display reverse order | O(n) |

## Would an ArrayList be a better choice? Why or why not?

Yes, **ArrayList is a better choice** for this comment section system because of

**Editing Efficiency**: O(1) vs. O(n)   
Explanation: For editing comment, ArrayList takes O(1) time complexity

i.e.: set (index, value) is O(1) with direct index access.