BERKELEY ALGORITHM

```
import java.util.*;

class Clock {
   int hours, minutes;

   Clock(int h, int m) {
      hours = h % 24;
      minutes = m % 60;
   }

   int getTotalMinutes() {
      return hours * 60 + minutes;
   }

   void adjustTime(int diff) {
      int total = getTotalMinutes() + diff;
      hours = (total / 60) % 24;
      minutes = total % 60;
   }

   String getTime() {
      return String.format("%02d:%02d", hours, minutes);
   }
}

public class BerkeleySimple {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      Clock[] clocks = new Clock[3];

      // Step 1: Input time for 3 clocks
      for (int i = 0; i < 3; i++) {
         System.out.print("Enter time for Clock " + (i + 1) + " (HH MM): ");
         int h = sc.nextInt();
         int m = sc.nextInt();
         clocks[i] = new Clock(h, m);
      }

      // Display initial times
      System.out.println("\nInitial Clock Times:");
      for (int i = 0; i < 3; i++) {
         System.out.println("Clock " + (i + 1) + (i == 0 ? " (master)" : "") + " -> " + clocks[i].getTime());
      }

      // Step 2: Show all times
      System.out.println("\nMaster requests time from all clocks:");
      for (int i = 0; i < 3; i++) {
         System.out.println("Clock " + (i + 1) + " -> " + clocks[i].getTime());
      }

      // Step 3: Calculate average time in minutes
      int total = 0;
      for (Clock c : clocks) total += c.getTotalMinutes();
      int average = total / 3;

      // Step 4: Show differences from master
      System.out.println("\nTime differences from master:");
      int masterTime = clocks[0].getTotalMinutes();
      for (int i = 1; i < 3; i++) {
         int diff = clocks[i].getTotalMinutes() - masterTime;
         System.out.printf("Clock %d: %+d minutes\n", i + 1, diff);
      }

      // Step 5: Adjust all clocks to average
      System.out.println("\nCorrected Clock Times:");
      for (int i = 0; i < 3; i++) {
         int diff = average - clocks[i].getTotalMinutes();
         clocks[i].adjustTime(diff);
         System.out.printf("Clock %d -> %s (%+d min)\n", i + 1, clocks[i].getTime(), diff);
      }

      sc.close();
   }
}
```

TOKEN RING

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.util.Arrays;

class TokenRing {
   private boolean hasToken = false;

   public synchronized void requestToken() {
      while (!hasToken) {
         try {
            wait();
         } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
         }
      }
   }

   public synchronized void giveToken() {
      hasToken = true;
      notify();
   }

   public synchronized void passToken() {
      hasToken = false;
      notify();
   }
}



public class TokenRingExample {
   private static int totalProcesses;
   private static int[] processIDs;
    private static TokenRing tokenRing = new
TokenRing();

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the total number of
processes: ");
      totalProcesses = scanner.nextInt();
      scanner.nextLine(); // Consume newline left
after nextInt()

      processIDs = new int[totalProcesses];

           System.out.println("Enter process IDs
separated by space:");
      for (int i = 0; i < totalProcesses; i++) {
         processIDs[i] = scanner.nextInt();
      }
       scanner.nextLine(); // Consume newline left
after loop

      Arrays.sort(processIDs);

      TokenPassing(scanner);
   }

     private static void TokenPassing(Scanner
scanner) {
      tokenRing.giveToken();

      int i = 0;
      while (i < totalProcesses) {
         int currentProcessID = processIDs[i];
         String theRing = "";
         int tokenIndex;

         System.out.print("Do you want to pass the
token to the next process? (y/n): ");
                       String passOrComplete =
scanner.nextLine();

                                                  if
(passOrComplete.equalsIgnoreCase("n")) {
            tokenIndex = (i + 2) % totalProcesses;
         } else {
            tokenIndex = (i + 1) % totalProcesses;
         }

         tokenRing.requestToken();
                   System.out.println("Process " +
currentProcessID + " is in the critical section.");

         try {
            Thread.sleep(1000);  // Simulate critical
section
         } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
         }

                                                  if
(passOrComplete.equalsIgnoreCase("n")) {
                    System.out.println("Process " +
currentProcessID + " completed and did not pass
the token.");
                                   }    else    if
(passOrComplete.equalsIgnoreCase("y")) {
            // Ask the user to enter a string to write
to the shared file
                 System.out.print("Enter a string to
append to the log file for Process " +
currentProcessID + ": ");
             String userInput = scanner.nextLine();  //
Now reads full input including after Enter
```

```
                // Append the string to a single shared
file
                        try (FileWriter writer = new
FileWriter("process_log.txt", true)) {
                            writer.write("Process " +
currentProcessID + ": " + userInput + "\n");
                System.out.println("String appended
to process_log.txt");
            } catch (IOException e) {
                        System.out.println("An error
occurred while writing to the file.");
            }

            // Pass the token to the next process
            int nextIndex = (i + 1) % totalProcesses;
                              int nextProcessID =
processIDs[nextIndex];
                    System.out.println("Process " +
currentProcessID + " passed the token to Process
" + nextProcessID);
         } else {
                  System.out.println("Invalid input,
please enter 'y' or 'n'.");
            continue;
         }

         System.out.println(theRing);

         tokenRing.passToken();
         tokenRing.giveToken();

         try {
              Thread.sleep(1000);  // Simulate token
passing delay
         } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
         }

         i++;
      }
   }
}
```

MAIN.JAVA (RING 6B)

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter number of processes in ring: ");
      int n = sc.nextInt();
      sc.nextLine();  // Consume newline
      List<Process> ring = new ArrayList<>();
      for (int i = 0; i < n; i++) {
         ring.add(new Process(i)); }
      // Set static ring reference
      Process.ring = ring;
      System.out.print("Enter IDs of crashed processes (space separated, press Enter to finish): ");
      String[] crashedIds = sc.nextLine().split("\\s+");
      for (String id : crashedIds) {
         if (!id.isEmpty()) {
            int crashedId = Integer.parseInt(id);
            if (crashedId >= 0 && crashedId < n) {
               ring.get(crashedId).isAlive = false;
               System.out.println("Process " + crashedId + " is marked as crashed."); }}}
      System.out.print("\nEnter ID of process to start election: ");
      int starter = sc.nextInt();
      if (starter < 0 || starter >= n) {
        System.out.println("Invalid process ID.");
      } else if (!ring.get(starter).isAlive) {
         System.out.println("Cannot start election from a crashed process.");
      } else {
         ring.get(starter).startElection(); }
      sc.close();}}
```

PROCESS.JAVA

```
import java.util.*;
public class Process {
   int id;
   boolean isAlive;
   static List<Process> ring;

   public Process(int id) {
      this.id = id;
      this.isAlive = true;
   }
   public void startElection() {
      System.out.println("\nProcess " + id + " starts an election.");
      List<Integer> electionPath = new ArrayList<>();
      Set<Integer> candidates = new HashSet<>();
      int n = ring.size();
      int current = (this.id + 1) % n;

      electionPath.add(this.id);
      if (this.isAlive) {
         candidates.add(this.id);
      }

      int sender = this.id;

      while (current != this.id) {
         Process receiver = ring.get(current);
         electionPath.add(receiver.id);

         if (receiver.isAlive) {
            System.out.println("Process " + sender + " -> ELECTION -> Process " + receiver.id);
            candidates.add(receiver.id);
         } else {
            System.out.println("Process " + sender + " -> ELECTION -> Process " + receiver.id + " (crashed, no response)");
         }

            System.out.println("Election path: " + electionPath + "\n");

         sender = receiver.id;
         current = (current + 1) % n;
      }

      int newCoordinator = Collections.max(candidates);
      System.out.println("Final Election path: " + electionPath);
      System.out.println("Election complete. Process " + newCoordinator + " is elected as coordinator.");
      announceCoordinator(newCoordinator);
   }

   public void announceCoordinator(int coordinatorId) {
      System.out.println("\nCoordinator Announcement:");

      for (Process p : ring) {
         if (p.id != coordinatorId && p.isAlive) {
            System.out.println("Process " + coordinatorId + " (Coordinator) -> COORDINATOR -> Process " + p.id);
         }}
      System.out.println("\nProcess " + coordinatorId + " is now the coordinator.");
   }
}
```

BULLYALGORITHMSIMULATION.JAVA

```
import java.util.*;

public class BullyAlgorithmSimulation {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      // Step 1: Number of processes
      System.out.print("Enter number of processes: ");
      int n = scanner.nextInt();

      List<Process> processes = new ArrayList<>();
      for (int i = 1; i <= n; i++) {
         processes.add(new Process(i));
      }

      for (Process p : processes) {
         p.setProcesses(processes);
      }

      // Step 2: Crashed process
      int crashedId = -1;
      boolean validCrashId = false;
      while (!validCrashId) {
         System.out.print("Enter the ID of the crashed process: ");
         crashedId = scanner.nextInt();
         if (crashedId < 1 || crashedId > n) {
            System.out.println("Invalid process ID. Please try again.");
         } else {
            validCrashId = true;
         }
      }

      processes.get(crashedId - 1).isAlive = false;
      System.out.println("Process " + crashedId + " has crashed.\n");

      // Step 3: Election initiator
      int starterId = -1;
      boolean validStarterId = false;
      while (!validStarterId) {
         System.out.print("Enter the ID of the process to start the election: ");
         starterId = scanner.nextInt();
         if (starterId < 1 || starterId > n) {
            System.out.println("Invalid process ID. Please try again.");
         } else if (!processes.get(starterId - 1).isAlive) {
            System.out.println("Process " + starterId + " is crashed or not alive. Choose a different process.");
         } else {
            validStarterId = true;
         }
      }

      // Start the election from the chosen process
      processes.get(starterId - 1).startElection();
   }
}
```

PROCESS.JAVA

```
import java.util.*;

public class Process {
   int id;  // Process ID
   boolean isAlive;  // Process state (alive or crashed)
   boolean isCoordinator;  // Indicates if this process is the coordinator
   boolean electionStarted = false;  // To prevent multiple elections for the same process
   List<Process> processes;  // List of all processes

   // Constructor to initialize the process
   public Process(int id) {
      this.id = id;
      this.isAlive = true;  // By default, the process is alive
      this.isCoordinator = false;  // Initially not the coordinator
   }

   // Method to set the list of all processes (used for election communication)
   public void setProcesses(List<Process> processes) {
      this.processes = processes;
   }

   // Method to start the election
   public void startElection() {
      // If the election has already started or the process is not alive, skip
      if (electionStarted || !isAlive) return;

      electionStarted = true;  // Mark that this process has started the election
      System.out.println("\nProcess " + id + " starts an election.");
```

```
        List<Process> higherProcesses = new
ArrayList<>();  // Higher priority processes (with
higher ID)
        List<Process> aliveResponders = new
ArrayList<>();  // Alive processes that respond
with OK

      // Step 1: Send ELECTION message to all
higher processes
      for (Process p : processes) {
         if (p.id > this.id) {
            System.out.println("Process " + id + " -
> ELECTION -> Process " + p.id);
            higherProcesses.add(p);
         }
      }

     // Step 2: Wait for OK responses from higher
processes
      for (Process p : higherProcesses) {
         if (p.isAlive) {
             System.out.println("Process " + p.id +
" -> OK -> Process " + id);
            aliveResponders.add(p);
         } else {
             System.out.println("Process " + p.id +
" is crashed. No OK sent.");}}
       // Step 3: Recursively start elections for
processes that respond with OK
      for (Process p : aliveResponders) {
         p.startElection();}
     // Step 4: If no higher process is alive, become
coordinator
      if (aliveResponders.isEmpty()) {
         becomeCoordinator();}}
     // Method to mark this process as the
coordinator
   public void becomeCoordinator() {
      // Only become coordinator once and ensure
the process is alive
      if (!isAlive || isCoordinator) return;

      isCoordinator = true;  // Mark the process as
the coordinator
        System.out.println("\nProcess " + id + "
becomes the coordinator.");

      // Inform all other alive processes about the
new coordinator
      for (Process p : processes) {
         if (p.id != this.id && p.isAlive) {
            System.out.println("Process " + id + " -
> COORDINATOR -> Process " + p.id);}}}}
```

RMI

CLIENT.JAVA

```
import java.rmi.*;
import java.util.Scanner;
public class Client {
   public static void main(String[] args){
      Scanner sc=new Scanner(System.in);
      try{
         String url="rmi://localhost/Server";
                                          ServerIntf
s=(ServerIntf)Naming.lookup(url);
         System.out.println("Enter num1:");
         int a=sc.nextInt();
         System.out.println("Enter num2:");
         int b=sc.nextInt();
         sc.nextLine();
         System.out.println("Enter str1:");
         String str1=sc.nextLine();
         System.out.println("Enter str2:");
         String str2=sc.nextLine();

                              System.out.println("Add
is:"+s.addition(a,b));
                         System.out.println("subtract
is:"+s.subtract(a,b))
                  System.out.println("multiplication
is:"+s.multiplication(a,b));
                          System.out.println("division
is:"+s.division(a,b));
                            System.out.println("square
is:"+s.square(a));
                    System.out.println("square   root
is:"+s.squareroot(b));

         System.out.println("Palindrome of string
is:"+s.palindrome(str1));

            System.out.println("String is equal or
not:"+s.isequalstring(str1,str2));

      }
      catch(Exception e){
                System.out.println("Exception   at
client"+e); }
     sc.close();}}
```

SERVER.JAVA

```
import java.rmi.*;
public class Server{
   public static void main(String[]args){
      try{
        ServerImpl serverimpl=new ServerImpl();
         Naming.rebind("Server",serverimpl);
         System.out.println("Server Started!!");
      }
      catch(Exception e){
          System.out.println("Exception occured at server!"+ e.getMessage());
      }}}
```

SERVERIMPL.JAVA

```
import java.rmi.*;
import java.rmi.server.*;
public class ServerImpl extends UnicastRemoteObject implements ServerIntf{
   public ServerImpl()throws RemoteException{

   }
     public int addition(int a,int b)throws RemoteException{
      return a+b;
   }
     public int subtract(int a,int b)throws RemoteException{
      return a-b;
   }
    public int multiplication(int a,int b)throws RemoteException{
      return a*b;
   }
     public int division(int a,int b)throws RemoteException{
      return a/b;
   }
         public int square(int a)throws RemoteException{
      return a*a;
   }
       public float squareroot(int a)throws RemoteException{
      return (float) (Math.sqrt(a));

   }
    public String palindrome(String str) throws RemoteException{
      StringBuilder sb=new StringBuilder(str);
      sb.reverse();
      if(str.equals(sb.toString())){
          return "it is palindrome";
      }
      else{
          return "not palindrome";
      }
   }

    public String isequalstring(String str1,String str2) throws RemoteException{
      if(str1.equals(str2)){
         return "string are equal";
      }
      else{
         return "string are not equal";}}}
```

SERVERINTF.JAVA

```
import java.rmi.*;
interface ServerIntf extends Remote{
       public int addition(int a,int b)throws RemoteException;
       public int subtract(int a,int b)throws RemoteException;
     public int multiplication(int a,int b)throws RemoteException;
       public int division(int a,int b)throws RemoteException;
            public int square(int a)throws RemoteException;
        public float squareroot(int a)throws RemoteException;
     public String palindrome(String str)throws RemoteException;
    public String isequalstring(String str1,String str2)throws RemoteException; }
```

MPI

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[])
 {
 int rank, size;
 int N = 16;
 int array[N];
 int local_sum = 0, total_sum = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD,
&rank);
 MPI_Comm_size(MPI_COMM_WORLD,
&size);
 int elements_per_proc = N / size;
 int local_array[elements_per_proc];
 if (rank == 0)
 {
 for (int i = 0; i < N; i++)
 {
 array[i] = i + 1;
 }
 printf("Original array: ");
 for (int i = 0; i < N; i++)
 {
 printf("%d ", array[i]);
 }
 printf("\n");
 }
 MPI_Scatter(array,              elements_per_proc,
MPI_INT, local_array, elements_per_proc,
MPI_INT, 0, MPI_COMM_WORLD);

 for (int i = 0; i < elements_per_proc; i++)
 {
 local_sum += local_array[i];
 }
 printf("Processor   %d   calculated   local   sum:
%d\n", rank, local_sum);

 MPI_Reduce(&local_sum,      &total_sum,      1,
MPI_INT,                  MPI_SUM,                  0,
MPI_COMM_WORLD);
 if (rank == 0)
 {
 printf("Total sum of array: %d\n", total_sum);
 }
 MPI_Finalize();
 return 0;
}
```

COMMANDS:

```
mpicc filename.c -o myexe
mpirun  --oversubscribe -np 11 myexe
```

CORBA

PALINDROMEMODULE.IDL

```
module PalindromeModule {
   interface Palindrome {
      boolean isPalindrome(in string input);
   };
};
```

PALINDROMESERVER.JAVA

```
import PalindromeModule.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;
import
org.omg.CosNaming.NamingContextPackage.*;

class PalindromeImpl extends PalindromePOA {
   public boolean isPalindrome(String input) {
      String reversed = new
StringBuilder(input).reverse().toString();
      return input.equalsIgnoreCase(reversed);
   }
}

public class PalindromeServer {
   public static void main(String[] args) {
      try {
         ORB orb = ORB.init(args, null);
         POA rootpoa =
POAHelper.narrow(orb.resolve_initial_referenc
es("RootPOA"));
         rootpoa.the_POAManager().activate();

         PalindromeImpl palindromeImpl = new
PalindromeImpl();
         org.omg.CORBA.Object ref =
rootpoa.servant_to_reference(palindromeImpl);
         Palindrome palindromeRef =
PalindromeHelper.narrow(ref);

         org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
         NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);

         NameComponent[] path =
ncRef.to_name("Palindrome");
         ncRef.rebind(path, palindromeRef);

         System.out.println("Palindrome Server
Ready...");
         orb.run();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

PALINDROMECLIENT.JAVA

```
import PalindromeModule.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import java.util.Scanner;

public class PalindromeClient {
   public static void main(String[] args) {
      try {
         ORB orb = ORB.init(args, null);
         // Obtain a reference to the naming service
                 org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
                      NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);
        // Resolve the reference to the remote object
                    Palindrome palindromeRef =
PalindromeHelper.narrow(ncRef.resolve_str("P
alindrome"));

         // Take input from user
                         Scanner scanner = new
Scanner(System.in);
         System.out.print("Enter a string to check
for palindrome: ");
         String input = scanner.nextLine();
         scanner.close();

         // Call the remote method
                              boolean result =
palindromeRef.isPalindrome(input);

         // Print the result
         System.out.println("Is \"" + input + "\" a
palindrome? " + result);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```