Matthew A. Krieger

November 17, 2015

CS 253: Data & File Structures

Homework IV

Neli P. Zlatareva, Ph.D.

Homework IV

**Problem I:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sorting Methods | | | | | |
|  | Time Complexity: | | | Stable? | Notes: |
|  | Best: | Worst: | Average: |
| Bubble Sort | O(n) | O(n2) | O(n2) | yes | Straightforward, Simple, Slow |
| Insertion Sort | O(n) | O(n2) | O(n2) | yes | Efficient for small list, sorts big arrays slowly, save memory |
| Selection Sort | O(n2) | O(n2) | O(n2) | No | Usually unstable but can be stable, slow for lots of data |
| **Quick Sort** | **O(n log n)** | **O(n2)** | **O(1.38 n log n)** | **No** | Best case: when pivot divides equal halves  Worst Case: Array already sorted – 1 / n-1 partition |
| **Shell Sort** | **O(n log n)** | **O(n2)** | **Depends on gap sequence** | **No** | Small code size, no use of call stack, reasonably fast, useful where memory is at a premium such as embedded and older mainframe applications. |
| **Heap Sort** | **O(n log n)** | **O(n log n)** | **O(n log n)** | **No** |  |

Heap Sort:

|  |  |  |
| --- | --- | --- |
| Heap Sort | | |
| Case: | Comparisons: | Exchanges: |
| Best Case: | 3591 | 2009 |
| Worst Case: | 1999 | 0 |
| Average Case: | 3202 | 1485 |
| Trial Run #1: | 3213 | 1506 |
| Trial Run #2: | 3143 | 1467 |
| Trial Run #3: | 3230 | 1481 |
| Trial Run #4: | 3229 | 1494 |
| Trial Run #5: | 3194 | 1477 |

**Best Case: O(n log n)**

**Average Case: O(n log n)**

**Worst Case: O(n log n)**

Shell Sort:

|  |  |  |
| --- | --- | --- |
| Shell Sort | | |
| Case: | Comparisons: | Exchanges: |
| Best Case: | 18,165 | 0 |
| Worst Case: | 18,165 | 7596 |
| Average Case: | 18,165 | 13,622 |
| Trial Run #1: | 18,165 | 13617 |
| Trial Run #2: | 18,165 | 13516 |
| Trial Run #3: | 18,165 | 13665 |
| Trial Run #4: | 18,165 | 13501 |
| Trial Run #5: | 18,165 | 13813 |

**Best Case: O(n log n)**

**Average Case: Depends on gap sequence**

**Worst Case: O(n2)**

Quick Sort:

|  |  |  |
| --- | --- | --- |
| Quick Sort | | |
| Case: | Comparisons: | Exchanges: |
| Best Case: | 6,901 | 0 |
| Worst Case: | 1999000 | 1000 |
| Average Case: | 9174 | 1754 |
| Trial Run #1: | 9157 | 1792 |
| Trial Run #2: | 9173 | 1787 |
| Trial Run #3: | 9169 | 1754 |
| Trial Run #4: | 9181 | 1743 |
| Trial Run #5: | 9190 | 1694 |

**Best Case: O(n log n)**

**Average Case: O(1.38n log n)**

**Worst Case: O(n2)**

Source Code:

**AnyType <<interface>>**

|  |
| --- |
| import java.util.Scanner;  interface AnyType  {  public boolean isBetterThan(AnyType datum);  public boolean isLessThan(AnyType datum);  } |

**StringType:**

|  |
| --- |
| class StringType implements AnyType {    private String word;  public StringType(){  word = "";  }  public StringType(String s){  word = s;  }  public boolean isBetterThan(AnyType datum) {  return (this.word.compareTo(((StringType)datum).word) > 0);  }  public boolean isLessThan(AnyType datum) {  return (this.word.compareTo(((StringType)datum).word) < 0);  }  public String toString() {  return word;  }  } |

**IntegerType:**

|  |
| --- |
| class IntegerType implements AnyType {  private int number;  public IntegerType() {  number = 0;  }    public IntegerType(int i) {  number = i;  }  public boolean isBetterThan(AnyType datum) {  return (this.number > ((IntegerType)datum).number);  }    public boolean isLessThan(AnyType datum) {  return (this.number < ((IntegerType)datum).number);  }  public int toInteger() {  return number;  }  } |

**Sort:**

|  |
| --- |
| import java.util.\*;  import java.io.\*;  class Sort  {    private int exchanges = 0;  private int compares = 0;    public static void bubbleSortAscending(AnyType[] array)  {  AnyType temp;  int numberOfItems = array.length;  boolean cont = true;  int comparisons = 0;  int exchanges = 0;    for (int pass=1; pass != numberOfItems; pass++)  {  if (cont)  {  cont = false;  for (int index=0; index != numberOfItems-pass; index++)  {  comparisons++;  if (array[index].isBetterThan(array[index+1]))  {  temp = array[index];  array[index] = array[index+1];  array[index+1] = temp;  exchanges++;  cont = true;  } // end inner if  } // end inner for  }    else  break; // end outer if  }    System.out.println("Number of comparisons: " + comparisons);  System.out.println("Number of exchanges: " + exchanges);  }    public static void bubbleSortDescending(AnyType[] array) {  AnyType temp;  int numberOfItems = array.length;  int comparisons = 0;  int exchanges = 0;  for (int pass=1; pass != numberOfItems; pass++) {  int count = 0;  for (int index=0; index != numberOfItems-pass; index++) {  comparisons++;  if (array[index].isLessThan(array[index+1])) {  temp = array[index];  array[index] = array[index+1];  array[index+1] = temp;  exchanges++;  count++;  }  }  if (count == 0)  {  break;  }  }  System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }    public static void insertionSortAscending(AnyType[] array) {  AnyType temp;  int comparisons = 0;  int exchanges = 0;  for (int i=0; i < array.length; i++) {    AnyType v;  v=array[i];  int j;  for (j = i - 1; j >= 0; j--) {  comparisons++;  if (v.isBetterThan(array[j])) break;  array[j + 1] = array[j];  exchanges++;  }  array[j + 1] = v;  }  System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }  // INSERTION SORT DESCENDING  public static void insertionSortDescending(AnyType[] array) {  AnyType temp;  int comparisons = 0;  int exchanges = 0;  for (int j = 1; j < array.length; j++)  {  temp = array[j];  int i = j;  if (i > 0 && array[i-1].isLessThan(temp))  {  do  {  array[i] = array[i-1];  exchanges++;  comparisons++;  i--;  } while (i > 0 && array[i-1].isLessThan(temp));  }  else if (i > 0 && array[i-1].isBetterThan(temp))  {  comparisons++;  }  array[i] = temp;  }  System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }        // SELECTION SORT  public static void selectionSortAscending(AnyType[] array)  {  int comparisons = 0;  int exchanges = 0;  for (int i = 0; i < array.length-1 ; i++)  {  int min = i;  int count = 0;  for (int j=i+1;j<array.length;j++)  {  comparisons++;  if(array[min].isBetterThan(array[j]))  {  min=j;  }  }  if (min != i)  {  AnyType temp = array[i];  array[i]=array[min];  array[min]=temp;  exchanges++;  count++;  }  }  System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }  // SELECTION SORT DESCENDING  public static void selectionSortDescending(AnyType[] array)  {  int comparisons = 0;  int exchanges = 0;  for (int i = 0; i < array.length-1 ; i++){  int min = i;  for (int j=i+1;j<array.length;j++)  {  comparisons++;  if(array[min].isLessThan(array[j]))  {  min=j;  }  }  if (min != i)  {  AnyType temp = array[i];  array[i]=array[min];  array[min]=temp;  exchanges++;  }  }  System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }      // SHELL SORT  public static void shellSortAscending(AnyType[] array)  {  int comparisons = 0;  int exchanges = 0;  int increment = array.length/2;  int j;    while (increment > 0)  {  for (int i = increment; i < array.length; i++)  {  AnyType temp = array[i];  comparisons++;  for(j = i; j >= increment && array[j-increment].isBetterThan(temp); j -= increment)  {  array[j] = array[j-increment];  exchanges++;  }  array[j] = temp;  }  increment = (int) Math.round(increment/2.2);  }    System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }  // SHELL SORT DESCENDING  public static void shellSortDescending(AnyType[] array)  {  int comparisons = 0;  int exchanges = 0;  int increment = array.length/2;  int j;  while (increment > 0)  {  int count = 0;  for (int i = increment; i < array.length; i++)  {  AnyType temp = array[i];  comparisons++;  for(j = i; j >= increment && array[j-increment].isLessThan(temp); j -= increment)  {  array[j] = array[j-increment];  exchanges++;  }  array[j] = temp;  }  increment = (int) Math.round(increment/2.2);  }    System.out.println("Number of comparisons: "+ comparisons);  System.out.println("Number of exchanges: "+ exchanges);  }      // QUICK SORT  public static void quickSortAscending(AnyType[] array, int low, int high)  {  int i = low, j = high, comparisons = 0, exchanges = 0;  AnyType temp, pivot = array[(low + high)/2];  while (i <= j)  {  while(array[i].isLessThan(pivot))  {  i++;  comparisons++;  }  while(array[j].isBetterThan(pivot))  {  j--;  comparisons++;  }  if (i <= j) {  temp = array[i];  array[i] = array[j];  array[j] = temp;  i++;  j--;  exchanges++;  }  }  if (low < j)  {  quickSortAscending(array, low, j);  }  if (i < high)  {  quickSortAscending(array, i, high);  }      }    // QUICK SORT DESCENDING  public static void quickSortDescending(AnyType[] array, int low, int high)  {  int i = low, j = high, comparisons = 0, exchanges = 0;  AnyType temp, pivot = array[(low + high)/2];  while (i <= j)  {  while(array[i].isBetterThan(pivot))  {  i++;  comparisons++;  }  while(array[j].isLessThan(pivot))  {  j--;  comparisons++;  }  if (i <= j) {  temp = array[i];  array[i] = array[j];  array[j] = temp;  i++;  j--;  exchanges++;  }  }  if (low < j)  {  quickSortDescending(array, low, j);  }  if (i < high)  {  quickSortDescending(array, i, high);  }  }      } |

**TestSort:**

|  |
| --- |
| import java.io.\*;  import java.util.\*;  import java.util.Scanner;  public class TestSort  {  public static void main (String[] args) throws IOException  {  //declarations  final int size = 2000;  int orderOfData = 0;  int randomNumber = 0;  Scanner scan = new Scanner(System.in);  IntegerType[] data = new IntegerType[size];      while (orderOfData == 0)  {  //asks user to select the order relationship  System.out.println("1. Ascending Order");  System.out.println("2. Descending Order");  System.out.println("\nSelect The Number Of The Ordering Relationship You Wish To Use:");  orderOfData = scan.nextInt();  System.out.println();  System.out.println();    //Ascending Order Selected  if (orderOfData == 1)  {  int caseSelection = 0;  while (caseSelection == 0)  {  //asks user which case they want to use  System.out.println("1. Best Case");  System.out.println("2. Average Case");  System.out.println("3. Worst Case");  System.out.println("\nSelect The Number Of The Case You Wish To Use:");  caseSelection = scan.nextInt();  System.out.println();  System.out.println();    //Constructs the .txt file for Best Case  if (caseSelection == 1)  {  //writes to .txt file 1==>2000  PrintWriter out = new PrintWriter (new FileWriter ("BestCase.txt"));  for (int i = 0; i < size; i++)  {  out.println(i+1);  }  out.close();    BufferedReader s = new BufferedReader(new FileReader ("BestCase.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = new IntegerType(Integer.parseInt(d));  }  s.close();    }    //Constructs the .txt file for Average Case  else if (caseSelection == 2)  {  //writes to .txt file random numbers between 1 ==> 2000  PrintWriter out = new PrintWriter (new FileWriter ("AverageCase.txt"));  for (int i = 0; i < size; i++)  {  randomNumber = (int) (Math.random() \* 2000);  out.println(randomNumber);  }  out.close();    BufferedReader s = new BufferedReader(new FileReader ("AverageCase.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = new IntegerType(Integer.parseInt(d));  }  s.close();  }    //Constructs the .txt file for Worst Case  else if (caseSelection == 3)  {  //writes to .txt file 2000==>1  PrintWriter out = new PrintWriter (new FileWriter ("WorstCase.txt"));  for (int i = 0; i < size; i++)  {  out.println(size-(i));  }  out.close();    BufferedReader s = new BufferedReader(new FileReader("WorstCase.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = new IntegerType(Integer.parseInt(d));  }  s.close();  }    //user input is invalid  else  {  System.out.println("Invalid Selection. Try Again");  caseSelection = 0;  }  }    int sortStyle = 0;  while (sortStyle == 0)  {  //select desired sorting technique  System.out.println("1. Bubble Sort");  System.out.println("2. Insertion Sort");  System.out.println("3. Selection Sort");  System.out.println("4. Quick Sort");  System.out.println("5. Shell Sort");  System.out.println("\nPlease Enter The Number Of The Sort You Wish To Use");  sortStyle = scan.nextInt();  System.out.println();  System.out.println();    //Bubble Sort - Ascending Order  if (sortStyle == 1)  {  Sort.bubbleSortAscending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nBubble Sort Completed Succesfully");  System.out.println("Data Arraged In Ascending Order\n\n");  }    //Insertion Sort - Ascending Order  else if (sortStyle == 2)  {  Sort.insertionSortAscending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nInsertion Sort Completed Succesfully");  System.out.println("Data Arraged In Ascending Order\n\n");  }    //Selection Sort - Ascending Order  else if (sortStyle == 3)  {  Sort.selectionSortAscending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nSelection Sort Completed Succesfully");  System.out.println("Data Arraged In Ascending Order\n\n");  }    //Quick Sort - Ascending Order  else if (sortStyle == 4)  {  Sort.quickSortAscending(data, 0, data.length - 1);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("Quick Sort Completed Succesfully");  System.out.println("Data Arraged In Ascending Order\n\n");  }    //Shell Sort - Ascending Order  else if (sortStyle == 5)  {  Sort.shellSortAscending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("Shell Sort Completed Succesfully");  System.out.println("Data Arraged In Ascending Order\n\n");  }  //notifies user that their input is invalid  else  {  System.out.println("\nInvalid Selection. Try Again\n\n");  caseSelection = 0;  }  }  }    //Descending Order Selected  else if (orderOfData == 2)  {  int caseSelection = 0;  while (caseSelection == 0)  {  //asks user which case they want to use  System.out.println("1. Best Case");  System.out.println("2. Average Case");  System.out.println("3. Worst Case");  System.out.println("\nSelect The Number Of The Case You Wish To Use");  caseSelection = scan.nextInt();  System.out.println();  System.out.println();    //Constructs the .txt file for Best Case  if (caseSelection == 1)  {  PrintWriter out = new PrintWriter (new FileWriter ("BestCase.txt"));  for (int i = 0; i < size; i++)  {  out.println(size - (i));  }  out.close();    BufferedReader s = new BufferedReader(new FileReader("BestCase.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = new IntegerType(Integer.parseInt(d));  }  s.close();  }      //Constructs the .txt file for Average Case  else if (caseSelection == 2)  {  PrintWriter out = new PrintWriter (new FileWriter("AverageCase.txt"));  for (int i = 0; i < size+1; i++)  {  randomNumber = (int) (Math.random() \* 2000);  out.println(randomNumber);  }  out.close();    BufferedReader s = new BufferedReader(new FileReader("AverageCase.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = new IntegerType(Integer.parseInt(d));  }  s.close();  }    //Constructs the .txt file for Worst Case  else if (caseSelection == 3)  {  PrintWriter out = new PrintWriter (new FileWriter ("WorstCase.txt"));  for (int i = 0; i < size; i++)  {  out.println(i+1);  }  out.close();    BufferedReader s = new BufferedReader(new FileReader("WorstCase.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = new IntegerType(Integer.parseInt(d));  }  s.close();  }    //notifies user that their input is invalid  else  {  System.out.println("\nInvalid Selection. Try Again\n\n");  caseSelection = 0;  }  }    int sortStyle = 0;  while (sortStyle == 0)  {  System.out.println("1. Bubble Sort");  System.out.println("2. Insertion Sort");  System.out.println("3. Selection Sort");  System.out.println("\nPlease Enter The Number Of The Sort You Wish To Use");  sortStyle = scan.nextInt();  System.out.println();  System.out.println();    if (sortStyle == 1)  {  Sort.bubbleSortDescending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nBubble Sort Completed Succesfully");  System.out.println("Data Arraged In Descending Order\n\n");  }    else if (sortStyle == 2)  {  Sort.insertionSortDescending(data);  // PRINTS SORTED ARRAY TO NEW TEXT FILE  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nInsertion Sort Completed Succesfully");  System.out.println("Data Arraged In Descending Order\n\n");  }    else if (sortStyle == 3)  {  Sort.selectionSortDescending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nSelection Sort Completed Succesfully");  System.out.println("Data Arraged In Descending Order\n\n");  }    else if (sortStyle == 4)  {  Sort.quickSortDescending(data, 0, data.length-1);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nQuick Sort Completed Succesfully");  System.out.println("Data Arraged In Descending Order\n\n");  }    else if (sortStyle == 5)  {  Sort.shellSortDescending(data);  PrintWriter o = new PrintWriter (new FileWriter("Sorted.txt"));  for (int y = 0; y < data.length; y++)  {  o.println(data[y].toInteger());  }  o.close();    System.out.println("\nShell Sort Completed Succesfully");  System.out.println("Data Arraged In Descending Order\n\n");  }    else  {  System.out.println("\nInvalid Selection. Try Again\n\n");  sortStyle = 0;  }  }  }    else  {  System.out.println("\nInvalid Selection. Try Again\n\n");  orderOfData = 0;  }  }    /\*  \*\*\* In-Development: \*\*\*  System.out.println("1. Integer");  System.out.println("2. String");  System.out.println("Please enter the number of the data type that you wish to produce: ");  int dataType = scan.nextInt();  \*/  }  } |

**binarySearchR:**

|  |
| --- |
| import java.io.\*;  import java.util.\*;  public class binarySearchR  {  public static int binarySearchR(int[] sortedArray, int start, int end, int key)  {  if (start < end)  {  int mid = start + (end - start) / 2;  if (key < sortedArray[mid])  {  return binarySearchR(sortedArray, start, mid, key);  }  else if (key > sortedArray[mid])  {  return binarySearchR(sortedArray, mid+1, end , key);  }  else  {  return mid;  }  }  return -(start + 1);  }  public static void main(String[] args) throws IOException  {  int size = 2000, key = 1, randomNum;  Random generator = new Random();  int[] data = new int[size];  Scanner scan = new Scanner(System.in);  BufferedReader s = new BufferedReader(new FileReader("sorted.txt"));  for (int x = 0; x < data.length; x++)  {  String d = s.readLine();  data[x] = Integer.parseInt(d);  }  s.close();  while (key != -1)  {  System.out.println("Enter the number to be found (1-2000): ");  key = scan.nextInt();  int index = binarySearchR(data,0,data.length,key);  if (index >= 0)  {  System.out.println("Found "+key+" at "+index+" index");  }  else if (key == -1)  {  System.out.println("Program terminated.");  }  else  {  System.out.println(key+" was not found in this list.");  }  }  }  } |

**Factorial:**

|  |
| --- |
| import java.util.\*;  import java.io.\*;  public class Factorial {  public static void main (String[] args) {  int num, fact;  Scanner scan = new Scanner(System.in);  System.out.println("Factorial Program");  System.out.print("Enter a number: ");  num = scan.nextInt();  fact = fact(num);  System.out.println(num + "! = " + fact + ".");  }    static int fact(int n) {  // Base Case:  // If n <= 1 then n! = 1.  if (n <= 1) {  return 1;  }  // Recursive Case:  // If n > 1 then n! = n \* (n-1)!  else {  return n \* fact(n-1);  }  }  } |

**Fibonacci:**

|  |
| --- |
| import java.util.\*;  import java.io.\*;  public class Fibonacci  {  public static void main (String[] args)  {  Scanner scan = new Scanner(System.in);  int n = 0, fib;  while (n != -1)  {  System.out.println("Enter N: ");  n = scan.nextInt();  if (n == -1)  {  System.out.println("Program Finished.");  }  else  {  fib = Fibonacci(n);  System.out.println(fib);  System.out.println();  }  }  }  static int Fibonacci(int n)  {  if (n<=1)  {  return n;  }  else  {  return Fibonacci(n - 1) + Fibonacci(n - 2);  }  }  } |

**Problem II:**

Step I

**NULL**

Step II

100

Step III

90 100

Step IV

90

100

80

Step V

90

100

70 80

Step VII

70 90

100

80

60

Step VI

50 60

100

80

70 90