**MODUL IV**

**SORTING DAN SEARCHING**

* 1. **TUJUAN**

Tujuan dari praktikum ini adalah:

1. Mahasiswa dapat memahami algoritma *sorting* dan *searching*.
2. Mahasiswa dapat mengimplementasikan metode *sorting* dan metode *searching* dalam program.
   1. **DASAR TEORI**

**4.2.1*****Sorting***

*Sorting* adalah suatu proses pengurutan data yang sebelumnya disusun secara acak atau tidak teratur menjadi urut dan teratur menurut aturan tertentu. Ada dua teknik pengurutan data, yaitu *ascending* (pengurutan naik) dan *descending* (pengurutan turun) [5].

Jenis – jenis *sorting*:

1. *Bubble Sort*

*Bubble sort* merupakan cara pengurutan yang paling mudah diantara cara pengurutan data yang lainnya. Cara kerja dari *bubble sort* ini adalah dengan membandingkan elemen yang satu dengan elemen berikut. Berikut algoritma penyusunan *bubble sort*:

* + - * 1. Tentukan data-data yang akan diurutkan dan ditampung di dalam *array* (jika menggunakan *array*).
        2. Lakukan pemeriksaan kondisi:

1. Untuk pengurutan secara *ascending*, apabila data yang dibandingkan lebih besar dari data berikutnya, maka data akan diurutkan posisinya.
2. Untuk pengurutan secara *descending*, apabila data yang dibandingkan lebih kecil dari data berikutnya, maka posisi datanta ditukar.
   * + - 1. Ulangi langkah kedua sebanyak jumlah data yang ada.
         2. Ulangi langkah kedua sebagai perulangan bersarang dengan data yang dibandingkan merupakan data berikutnya, dan data yang dibandingkan adalah data terakhir.
         3. Tampilkan data yang telah terurut.
3. *Straight Selection Sort*

*Straight selection sort* adalah metode/cara pengurutan dengan membandingkan data yang satu dengan data-data berikutnya sampai dengan data terakhir. Jika ada ditemukan data yang lebih kecil, maka akan dicatat posisinya. Berikut algoritma penyusunan *straight selection sort*:

* + - * 1. Tentukan data-data yang akan diurutkan.
        2. Lakukan pemeriksaan kondisi:

1. Untuk pengurutan *ascending*, apabila ada ditemukan data yang lebih kecil dari data yang dibandingkan pada data-data setelahnya, maka posisi data yang lebih kecil tersebut dicatat dan ditukar posisinya.
2. Untuk pengurutan *descending*, apabila ada ditemukan data yang lebih besar dari data yang dibandingkan pada data-data setelahnya, maka posisi data yang lebih besar tersebut dicatat dan ditukar posisinya.
   * + - 1. Ulangi langkah kedua sampai semua data telah dibandingkan.
         2. Tampilkan data yang sudah diurutkan.
3. *Insertion sort*

*Insertion sort* merupakan suatu metode pengurutan yang membandingkan suatu elemen dengan elemen-elemen yang lain dan menyusunnya sedemikian rupa hingga elemen tersebut berada pada posisi yang seharusnya. Berikut algoritma penyusunan *insertion sort*:

* + - * 1. Tentukan data-data yang aka diurutkan dan disimpan dalam *array* (jika menggunakan *array*).
        2. Bandingkan data dengan data disebelah kirinya satu per satu.
        3. Lakukan pemeriksaan kondisi:

1. Untuk pengurutan secara *ascending*, apabila data yang dibandingkan lebih kecil dari data sebelah kirinya, maka sisipkan data yang dibandingkan disebelah kiri data yang lebih besar.
2. Untuk pengurutan secara *descending*, apabila data yang dibandingkan lebih besar dari data sebelah kirinya, maka sisipkan data yang dibandingkan disebelah kiri data yang lebih kecil.
   * + - 1. Ulangi langkah kedua hingga data yang dibandingkan dengan data yang paling kiri.
         2. Ulangi langkah kedua sebagai perulangan bersarang hingga semua data telah dibandingkan [6].
3. *Merge Sort*

*Merge sort* adalah *sort* yang dilakukan dengan teknik merge (menggabungkan) dua buah array kedalam sebuah array yang baru. Dimana prinsip utama yang diimplementasikan pada algoritma *merge-sort* seringkali disebut sebagai pecah-belah dan taklukkan (bahasa Inggris: *divide and conquer*). Hal ini dikarenakan algoritma sorting melakukan pembagian struktur data sebelum kemudian dioperasikan satu per satu. Berikut algoritma penyusunan *merge sort*:

1. Pertama kali *array* yang sudah diisi dibagi menjadi dua bagian.
2. Kedua *array* kemudian diurutkan secara terpisah pad masing-masing bagian sehingga menjadi berurut.
3. Sebuah *array* baru dibentuk yang sebagai penggabungan dari kedua *array* tersebut.
4. Langkah berikutnya adalah penggabungan dari masing-masing larik ke dalam larik baru yang dibuat sebelumnya [4].
5. *Quick Sort*

*Quick sort* merupakan suatu algoritma pengurutan data yang menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama partition exchange sort. Untuk memulai irterasi pengurutan, pertama-tama sebuah elemen dipilih dari data, kemudian elemen-elemen data akan diurutkan diatur sedemikian rupa. Metode *quick* sering disebut juga metode partisi (*partition exchange sort*). Berikut algoritma penyusunan *quick* sort:

1. Menentukan posisi kiri, kanan, pivot(tengah).
2. Posisi kiri bergerak ke kanan, posisi kanan bergerak ke kiri, berhenti jika data kiri lebih besar sama dengan data pivot dan data kanan lebih kecil sama dengan data pivot.
3. Tukar data kiri dan data kanan.
4. Pengecekan kiri dan kanan bertambah satu.
5. Melakukan kembali langkah b, berhenti bila *pointer* kiri *overlap* dengan *pointer* kanan.
6. Membagi menjadi dua bagian.
7. Setiap bagian diproses rekursif seperti langkah b sampai e.
8. Pembagian partisi berhenti bila tiap partisi hanya menyisakan satu elemen data saja.
9. Menggabungkan kembali partisi yang terdiri dari satu elemen jika sudah terurut [3].

**4.2.2 *Searching***

Algoritma pencarian (searching algorithm) adalah algoritma yang menerima sebuah argumen kunci dan dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut. Ada beberapa macam teknik pencarian seperti pencarian sekuensial dan pencarian biner. Perbedaan dari dua teknik ini terletak pada keadaan data. Pencarian sekuensial digunakan apabila data dalam keadaan acak atau tidak terurut (contoh: *sequential* *search*). Sebaliknya, pencarian biner digunakan pada data yang sudah dalam keadaan urut (contoh: *Binary* *serach* dan *interpolation* *search*)[3].

1. *Sequential* *Searching*

*Sequential* *Searching* merupakan pencarian data dengan prinsip data yang ada akan dibandingkan satu per satu secara berurutan dengan data yang dicari sampai data tersebut ditemukan atau tidak ditemukan. Pada dasarnya, pencarian ini hanya melakukan pengulangan dari 1 sampai dengan jumlah data. Pada setiap pengulangan, setiap data akan dibandingkan dengan yang dicari. Apabila sama, berarti data telah ditemukan, sebaliknya apabila sampai akhir pengulangan tidak ada data yang sama, berarti data tidak ditemukan. Pada kasus yang paling buruk, apabila terdapat n data maka harus dilakukan pencarian sebanyak n kali pula. Algoritmanya dapat dituliskan sebagai berikut:

|  |
| --- |
| i → Posisi data  i → Posisi data  x → Data yang di cari  n → Jumlah data yang tersedia  i → Posisi data  x → Data yang di cari  n → Jumlah data yang tersedia  x → Data yang di cari  n → Jumlah data yang tersedia |

1. i ← 0
2. Jika (data[i] = x) maka data ditemukan , melanjutkan ke langkah 3
3. Selama (i ≤ n) kerjakan langkah ke 4
4. i = i + 1 , melanjutkan langkah ke 5
5. Kembali ke langkah 2
6. *Binary* *Search*

Salah satu syarat agar pencarian biner dapat dilakukan adalah data yang tersedia sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, pencarian biner tidak dapat dilakukan. Algoritmanya dapat dituliskan sebagai berikut:

Inisialisasi awal :

Kiri = 0, kanan = n ( index terakhir dari array ), flag = 0

1. Mengecek apakah ( kiri <= kanan && flag 0 )
2. Jika ya, ke langkah 2
3. Jika tidak, ke langkah 6.
4. Menghitung nilai tengah = ( kiri + kanan ) / 2
5. Mengecek apakah cari = = data [ tengah ]
6. Jika ya, flag = 1. Ke langkah 6
7. Jika tidak, ke langkah 4
8. Mengecek apakah cari < data[tengah ]
9. Jika ya, kanan = tengah – 1. Ke langkah 1
10. Jika tidak ke langkah 5
11. Mengecek apakah cari > data [ tengah ]
12. Jika ya, kiri = tengah + 1. Ke langkah 1
13. Jika tidak, ke langkah 1
14. Mengecek apakah flag = = 1
15. Jika ya, data di temukan
16. Jika tidak, data tidak di temukan[1].
    1. **PERMASALAHAN**
17. Seorang pelatih sepak bola ingin mengatur strategi untuk memenangkan sebuah pertandingan final. Pelatih tersebut mempunyai 21 pemain, diantaranya 7 orang bek, 7 orang gelandang,dan 7 orang penyerang. Karena pada pertandingan semifinal sebelumnya strategi formasi yang digunakan oleh pelatih tersebut sudah dibaca oleh pihaklawan, oleh karenaitu pada pertandingan final, pelatihtersebut ingin merombak ulang susunan formasi dari timnya dengan melakukan rotasi pemain. Dimana pemain-pemain yang menjadi *stater*pada pertandingan tersebut adalah 11 orang dengan nomor punggung terkecil, sedangkan 10 orang lainnya akan menjadi cadangan. Namun sebelum melakukan rotasi pemain, pelatih tersebut ingin mendata semua pemainnya secara berurutan berdasarkan nomor punggung.
18. Bantu pelatih tersebut untuk mendata para pemainnya dengan menggunakan algoritma *insertion sort, selection sort,* dan *bubble sort.* Tentukan algoritma yang paling efisien.
19. Tampilkan pemain yang menjadi *starter*.

2. Setelah perang besar anatara aliansi Terran dan Protoss melawan Zerg usai dengan berhasil mengalahkan Zerg, pasukan gabungan Terran dan Protoss kembali ke pangakalan. Setelah sampai, kedua pasukan yang tersisa total berjumlah 30 denagn urutan acak (rentang angka 1 - 100). Kemudian barisan tersebut dibariskan kedalam dua barisan yaitu barisan Terran dan barisan Protoss (Terranmemiliki nomor antara 1 – 50 dan Protossmemiliki nomor antara 52 - 100 ). Setelah itu, kedua barisan tersebut dicek kondisi masing-masing pasukan akibat terkena parasite Zerg. Pada Terran pasukan yang terkena parasit adalah bilangan prima dan untuk Protoss adalah bilangan *Fibonacci*. Setelah ditemukan pasukan yang terkena parasit lalu dibariskan ke dalam barisan baru untuk mendapatkan perawatan. Tampilkan barisan Terran*,* Protoss, dan barisan yang akan mendapat perawatan serta tampilkan jumlah pencarian setiap pasukan yang terkena parasit.

* 1. **HASIL DAN PEMBAHASAN**
     1. **Program Pengaturan Pemain Sepak Bola**
        1. Algoritma
           1. *Bubble Sort*

1. Menentukan data-data yang akan diurutkan dan ditampung di dalam *array* (jika menggunakan *array*).
2. Melakukan pemeriksaan kondisi pengurutan secara *ascending*, apabila data yang dibandingkan Lebih besar dari data berikutnya, maka data akan diurutkan posisinya.
3. Mengulangi langkah kedua sebanyak jumlah data yang ada.
4. Mengulangi langkah kedua sebagai perulangan bersarang dengan data yang dibandingkan merupakan data berikutnya, dan data yang dibandingkan adalah data terakhir.
5. Menampilkan data yang telah terurut.
   * + - 1. *Insertion Sort*
6. Menentukan data-data yang aka diurutkan dan disimpan dalam *array* (jika menggunakan *array*).
7. Membandingkan data dengan data disebelah kirinya satu per satu.
8. Melakukan pemeriksaan kondisi pengurutan secara *ascending*, apabila data yang dibandingkan lebih kecil dari data sebelah kirinya, maka sisipkan data yang dibandingkan disebelah kiri data yang lebih besar.
9. Mengulangi langkah kedua hingga data yang dibandingkan dengan data yang paling kiri.
10. Mengulangi langkah kedua sebagai perulangan bersarang hingga semua data telah dibandingkan.
    * + - 1. *Selection Sort*
11. Menentukan data-data yang akan diurutkan.
12. Melakukan pemeriksaan kondisi pengurutan *ascending*, apabila ada ditemukan data yang lebih kecil dari data yang dibandingkan pada data-data setelahnya, maka posisi data yang lebih kecil tersebut dicatat dan ditukar posisinya.
13. Mengulangi langkah kedua sampai semua data telah dibandingkan.
14. Menampilkan data yang sudah diurutkan.
    * + 1. *Source code*

|  |
| --- |
| package jurnal;  import java.util.Random;  public class no1 {  node head, tail;  int bubble=0, straight=0, insertion=0;    class node{  int data;  node next,prev;  public node (int a){  data = a;  }  }    public void Add(int m){  node temp = new node(m);  if(head == null){  head = temp;  tail = temp;  }  else {  temp.next = head;  head.prev = temp;  head = temp;  }  }    public int SortB(){  node temp = head;  while(temp != null){  node temp1 = head;  node temp2 = temp1.next;  while(temp2 != null){  if(temp2.data > temp1.data){  int a=temp2.data;  temp2.data=temp1.data;  temp1.data=a;  bubble++;  }  temp1=temp1.next;  temp2=temp2.next;  }  temp=temp.next;  }  System.out.print("\nSwapping bubble = " + bubble +" kali | urutan = ");  return bubble;  }    public int SortS(){  node temp, temp2, bantu;  temp = head;  int s;  for(int i=0; i<10; i++) {  temp2 = temp.next;  straight++;  for(int j=i+1;j<10;j++) {  if(temp.data < temp2.data) {  s = temp2.data;  temp2.data = temp.data;  temp.data = s;  }  temp2 = temp2.next;  }  temp = temp.next;  }  System.out.print("Swapping selection = " + straight +" kali | urutan = ");  return straight;  }    public int SortI (){  node temp = head;  node prev = temp;  int store;  for(int i = 0;i<10-1; i++) {  insertion++;  node temp2 = temp.next;  for(int j = i+1; j>0;j--) {  if(prev.data<temp2.data) {  store = prev.data;  prev.data = temp2.data;  temp2.data = store;  temp2 = temp2.prev;  prev = prev.prev;  }  else  break;  }  temp=temp.next;  prev = temp;  }  System.out.print("Swapping Insertion = " + insertion +" kali | urutan = ");  return insertion;  }    public void Show (){  node temp = tail;  while (temp!=head.prev){  System.out.print("[" + temp.data + "] ");  temp=temp.prev;  }  }    public void keluar(){  node temp = head;  head = head.next;  temp.next = null;  head.prev = null;}  public static void main(String[] args) {  Random no = new Random();  no1 a = new no1();    for(int i = 1;i<=21;i++){  int put = no.nextInt(25);  a.Add(put);  }  System.out.println("Nomor-nomor punggung pemain: "); a.Show();  System.out.println();    int bub = a.SortB();  a.Show();  System.out.println();    int sel = a.SortS();  a.Show();  System.out.println();    int ins = a.SortI();  a.Show();  System.out.println();    System.out.print("\nPemain yang menjadi starter: ");  for(int i=1; i<=11; i++){  a.keluar();  }  a.Show();  System.out.println();    if(bub < sel){  if(bub < ins)  System.out.println("Jadi, Sorting yang terbaik adalah = Bubble Sort");  else  System.out.println("Jadi, Sorting yang terbaik adalah = Insertion Sort");  }  else{  if(sel < ins)  System.out.println("Jadi, Sorting yang terbaik adalah = Selection Sort");  else  System.out.println("Jadi, Sorting yang terbaik adalah = Insertion Sort");  }}} |

* + - 1. Hasil *run* program

|  |
| --- |
|  |

Gambar 4.1 Hasil *run* program pengaturan pemain sepak bola

Pada **Gambar 4.1** merupakan hasil *run* program dari pengaturan pemain sepak bola, dimana pada setiap pengaturan pemain digunakan *sorting* untuk menentukan pemain mana yang akan dijadikan *starter*, jadi pengaturan dengan *sorting* yang paling efisienlah yang akan digunakan.

* + 1. **Program Pasukan Terran dan Protoss**

1. Algoritma
2. Membuat metode untuk menambahkan data.
3. Memeriksa kondisi apakah program tersebut sudah ada datanya atau belum.
4. Saat program belum memiliki data maka data baru menjadi *head* dan *tail*nya.
5. Saat program sudah memiliki data maka data yang yang sudah ada *pointer*nya akan menunjuk ke data yang baru.
6. Membuat metode untuk menghapus *node* dalam *linked list*.
7. Saat data yang dicari berada di akhir *Linked list* maka *pointer*nya akan menunjuk ke data sebelumnya. dan *node* yang ada diakhir tadi dihapus.
8. Saat data yang dicari berada di awal *Linked list* maka *pointer*nya akan menunjuk ke data sesudahnya. dan *node* yang ada diawal tadi dihapus.
9. Saat data yang dicari tidak ada di akhir ataupun di awal maka *pointer*nya akan menunjuk ke data sebelumnya dan menunjuk kedirinya lagi yang hasil *pointer*nya akan menunjuk ke data setelahnya, dan juga *pointer* yang satu menunjuk ke data selanjutnya dan menunjuk ke dirinya lagi yang hasil *pointer*nya akan menunjuk ke data sebelumnya.
10. Membuat metode untuk mencari data bilangan prima.
11. Mendeklarasikan variabel yang digunakan untuk menampung banyak *factor* pemabagi data.
12. Melakukan perulangan dari 1 sampai banyak data yang dibuat.
13. Apabila jumlah *factor* pembaginya ada dua maka data tersebut bilangan prima.
14. Apabila jumlah *factor* pembaginya lebih dari dua maka data tersebut bukan bilangan prima.
15. Membuat metode untuk mencari bilangan *Fibonacci*.
16. Apabila data yang dicari menunjukkan data bilangan 55 dan 89 maka data tersebut merupakan bilangan *Fibonacci*.
17. Apabila data yang dicari tidak menunjukkan data bilangan 55 dan 89 maka data tersebut bukan merupakan bilangan *Fibonacci*.
18. *Source code*

|  |
| --- |
| package jurnal;  import java.util.Scanner;  import java.util.Random;  class node {  int data;  node next;  node prev;  node (int id){  this.data = id;  this.next = null;  this.prev = null;  }  }  public class no2 {  node head;  node tail;  int size=0;  void addLast(int d){  node baru = new node(d);  if ( head == null ){  head = tail = baru;  size++;  return;  }  tail.next = baru;  baru.prev = tail;  tail = baru;  size++;  }  void primTerr(no2 trr, no2 rawat){  node Terran = head;  while (Terran != null){  if ( prim(Terran.data) ){  rawat.addLast(Terran.data);  }  Terran = Terran.next;  }  }  static boolean prim(int d){  int sep = 0;  if ( d < 1 ){  return false;  }  for ( int i = 1; i<=d; i++ ){  if ( d%i == 0 ){  sep++;  }  }  if (sep == 2){  return true;  } else {  return false;  }  }  void fiboPros(no2 protosz, no2 rawat){  node protos = head;  while (protos != null){  if ( fibona(protos.data) ){  rawat.addLast(protos.data);  }  protos = protos.next;  }  }  static boolean fibona(int d){  int st = 0, nd = 1, fib = 0;  while( fib < d ){  fib = st+nd;  st = nd;  nd = fib;  }  if ( d == fib ){  return true;  }  else {  return false;  }  }    void display(){  node cur = head;  while (cur != null){  System.out.print(cur.data+" ");  cur = cur.next;  }  System.out.println("\n");  }    public static void main(String[] args) {  Random r = new Random();  no2 barisan = new no2();  no2 Terran = new no2();  no2 Protoss = new no2();  no2 perawatan = new no2();    for(int i=0; i<30; i++){  int masuk = r.nextInt(100);  barisan.addLast(masuk);  if (masuk<=50){  Terran.addLast(masuk);  } else if (masuk>50 || masuk<=100){  Protoss.addLast(masuk);  } i++;  }  System.out.println("Barisan Pasukan: ");  barisan.display();  System.out.println("Barisan Terran: ");  Terran.display();  System.out.println("Barisan Protoss: ");  Protoss.display();  Terran.primTerr(Terran, perawatan);  System.out.println("Pasukan Terran yang terkena parasi (bilangan prima): ");  perawatan.display();  Protoss.fiboPros(Protoss, perawatan);  System.out.println("Pasukan yang terkena parasit setelah ditambah Pasukan Protoss (bilangan fibonaci): ");  perawatan.display();  }  } |

1. Hasil *run* program

|  |
| --- |
|  |

Gambar 4.2 Hasil *run* program pasukan Terran dan Protoss

Pada **Gambar 4.2** merupakan hasil *run* dari program pasukan Terran dan Protoss, dimana pada gambar tersebut dapat dilihat bahwa ada barisan pasukan yang terdiri dari angka *random*, kemudian dari barisan pasukan tersebut dapat dilihat bahwa angka dibawah dari nilai 50 akan ditempatkan sebagai barisan Terran, kemudia yang terletak diatas nilai 50 akan ditempatkan sebagai barisan Protoss. Pasukan yang terkena parasit adalah pasukan Terran dengan angka prima kemudian pasukan Protoss yang terkena parasit merupakan bilangan *Fibonnaci* diatas 50.

* 1. **ANALISA**

**4.5.1`Program Pengaturan Pemain Sepak Bola**

|  |
| --- |
| import java.util.Random;  public class no1 {  node head, tail;  int bubble=0, straight=0, insertion=0; |

*Script* “import java.util.Random” adalah *header* pada java yang digunakan untuk membuar angka secara *random*. *Script* “public class no1” merupakan *class* utama yang digunakan fungsi *main*. Pada *code* “node head, tail;” dan “int bubble=0, straight=0, insertion=0;” merupakan pendeklarasian variabel.

|  |
| --- |
| class node{  int data;  node next,prev;  public node (int a){  data = a;  }  } |

*Code* “class node {” adalah pembuatan kelas bernama “node”, menampung pendeklarasian variabel “int data;”, “node next, prev;”. Pada *Code* “public node (int a)” merupakan sebuah konstruktor dari kelas “node” yang di dalamnya terdapat inisialisasi variabel “data=a;”.

|  |
| --- |
| public void Add (int m){  node temp = new node(m);  if(head==null){  head=temp;  tail=temp;  }else {  temp.next=head;  head.prev=temp;  head=temp;  }} |

*Code* “public void Add (int m){” merupakan sebuah *method* dengan nama “Add” dengan tipe data *void* memiliki parameter “int m”. *Method* ini berfungsi untuk menambah *node* dari belakang ke depan. *Script* “Jur1.node= new Jur1.node(m)” merupakan pembuatan objek *node* baru. Pada *script* “if(head==null)” merupakan sebuah kondisi yang mana jika kondisi terpenuhi maka akan melakukan proses inisialisasi variabel “head=temp; ” dan “tail=temp; ”, namun jika kondisi tidak terpenuhi maka akan melakukan proses “else” yang di dalamnya terdapat inisialisasi variabel pula “temp.next=head;”, “head.prev=temp;” dan “head=temp;”

|  |
| --- |
| Public int SortB (){  node temp = head;  while(temp!=null){  node temp1=head;  node temp2=temp1.next;  while(temp2!=null){  if(temp2.data<temp1.data){  int a=temp2.data;  temp2.data=temp1.data;  temp1.data=a;  bubble++;  }  temp1=temp1.next;  temp2=temp2.next;  }  temp=temp.next;  }  System.out.print("Swapping bubble = "+bubble+"kali | urutan = ");  return bubble;  } |

*Script* “public int SortB” adalah *method* bernama “SortB” yang digunakan untuk melakukan *bubble sort* atau *sorting* secara *linear* yaitu dengan mengecek satu demi satu *node*, “a=temp2.data” “temp2.data=temp1.data” “temp1.data=a” yang digunakan untuk memindahkan nilai ke *node* sebelumnya. “while(temp!=null){” apabila data tidak sama dengan *null* atau dalam artian lain *node* memiliki sebuah data yang ditampung, maka program akan menjalankan proses *looping* secara terus menerus hingga data selanjutnya bernilai *null* atau sudah tidak ada data lagi yang dapat dilalui, “if(temp2.data<temp1.data){” apabila nilai dari data setelah *node* sebelumnya lebih kecil, maka akan dilakukan pertukaran nilai data yang dimana nilai data yang lebih kecil akan ditempatkan di sisi kiri data yang memiliki nilai data yang lebih besar pada program tersebut.

|  |
| --- |
| public int SortS (){  node temp, temp2;  temp = head;  int s;  for(int i=0; i<10; i++) {  temp2 = temp.next;  straight++;  for(int j=i+1;j<10;j++) {  if(temp.data>temp2.data) {  s = temp2.data;  temp2.data = temp.data;  temp.data = s;  }  temp2 = temp2.next;  }  temp = temp.next;  }  System.out.print("Swapping selection = "+straight+"kali | urutan = ");  return straight;  } |

*Script* “public int SortS” adalah *method* bernama “SortS” yang digunakan untuk *selection sort* atau *sorting* dengan cara menyisipkan nilai dengan menukarnya. “temp2=temp.next” merupakan *pointer* penanda setelah *head*, yang digunakan untuk menunjuk *node* selanjutnya saat proses perulangan dilakukan. “for(int j=i+1;j<10;j++)” merupakan perulangan untuk mengulang sampai *size* dengan pengecekan “if(temp.data>temp2.data)” jika data setelahnya lebih besar maka akan mengerjakan kondisi yang didalam. “s=temp2.data”, “temp2.data = temp.data”, “temp.data = s”, merupakan *script* memindahkan nilai.

|  |
| --- |
| public int SortI (){  node temp = head;  node prev = temp;  int store;  for(int i = 0;i<10-1; i++) {  insertion++;  node temp2 = temp.next;  for(int j = i+1; j>0;j--) {  if(prev.data>temp2.data) {  store = prev.data;  prev.data = temp2.data;  temp2.data = store;  temp2 = temp2.prev;  prev = prev.prev;  }  else  break;  }  temp=temp.next;  prev = temp;  }  System.out.print("Swapping insertion = "+insertion+"kali | urutan = ");  return insertion;  } |

*Script* “public int SortI” *method* bernama “SortI” yang digunakan untuk mensorting nilai dengan cara *insertion sort* atau menyimpan nilai indeks, “store= prev.data”, “prev.data= temp2.data”, “temp2.data= store”, “temp2= temp2.prev”, “prev= prev.prev” yang digunakan untuk memindahkan nilai dengan cara menyimpan nilai yang akan dibandingkan dengan semua *node* kemudian memindahkan nilai ke *sorted list*. *Method* ini menggunakan prinsip kerja dari *double linked list*, dikarenakan pada *method* ini menggunakan *prev* dan *next* untuk melakukan penunjukkan suatu data, *prev* digunakan untuk menunjuk data sebelumnya sementara *next* digunakan untuk melakukan penunjukkan pada data selanjutnya. *Method* ini difungsikan untuk melakukan proses *sorting insertion*, dan pada tampilan akhirnya *method* ini akan menampilkan data yang telah di *sorting* dan berapa banyak iterasi yang dilakukan ketika proses *sorting* dilakukan.

|  |
| --- |
| public void Show (){  Jur1.node temp = tail;  while (temp!=head.prev){  System.out.print(“[” + temp.data + “] ”);  temp=temp.prev;  }  } |

*Script* “public void Show (){” untuk membuat sebuah *method* bernama “Show” yang digunakan untuk menampilkan semua isi dari *double linked list*. “while(temp!=head.prev)” digunakan untuk mengulang agar nilai data dapat bergeser kemudian “System.out.print(“[” + temp.data + “] ”);” akan ditampilkan secara horizontal dan “temp=temp.next” *script* menggeser *pointer*.

|  |
| --- |
| public static void main (String [] args){  no1 a = new no1();  Random r = new Random();    for(int i = 1;i<=10;i++){  int put = r.nextInt(100);  a.Add(put);  }  System.out.println("Nomor-nomor punggung pemain: "); a.Show();  System.out.println();    int c = a.SortB();  a.Show();  System.out.println();  int d = a.SortS();  a.Show();  System.out.println();  int e = a.SortI();  a.Show();  System.out.println();  System.out.print("\nPemain yang menjadi starter: ");  for(int i=1; i<=11; i++){  a.keluar();  }  a.Show();  System.out.println()    if(c<d){  if(c<e)  System.out.println("Jadi, Sorting yang terbaik adalah = Bubble Sort");  else  System.out.println("Jadi, Sorting yang terbaik adalah = Selection Sort");  }  else{  if(d<e)  System.out.println("Jadi, Sorting yang terbaik adalah = Insertion Sort");  else  System.out.println("Jadi, Sorting yang terbaik adalah = Selection Sort");  }}} |

*Script* bernama “public static void main (String[] args)” merupakan *method* *main* dimana digunakan untuk menaruh semua perintah yang akan dijalankan oleh *program*.“c=a.SortB()”, “d=a.SortS ()”, “e=a.SortI ()” merupakan pendeklarasian objek untuk memanggil *method*, “Random r=new Random()” yang digunakan untuk memanggil *method* sebagai objek “System.out.println()” kemudian menampilkan sesuai jurnal.

* + 1. **Program Pasukan Terran dan Protoss**

|  |
| --- |
| class node {  int data;  node next;  node prev;    public node (int data){  this.data=id;  next=null;  prev=null;  }  } |

*Code* “class node {” adalah pembuatan kelas bernama “node”, menampung pendeklarasian variabel “int data;”, “node next;” dan “node prev;”. Pada *Code* “public node (int data)” merupakan sebuah konstruktor dari kelas “node” yang di dalamnya terdapat inisialisasi variabel “this.data=data;”, “next=null;” dan “prev=null;”

|  |
| --- |
| public class no2 {  node head;  node tail;  int size=0; |

*Code* “public class no2 {” merupakan kelas utama bernama “no2”, dimana nantinya terdapat *method main* serta *method-method* lainnya dideklarasikan di dalamnya. Pada *code* “node head;”, “node tail”, “int size=0” merupakan pendeklarasian variabel.

|  |
| --- |
| void addLast(int d){  node baru = new node(d);  if ( head == null ){  head = tail = baru;  size++;  return;  }  tail.next = baru;  baru.prev = tail;  tail = baru;  size++;  } |

*Code* “public void addLast (int d){” merupakan sebuah *method* dengan nama “addLast” dengan tipe data *void* memiliki parameter “int d”. *Method* ini berfungsi untuk menambah *node* dari belakang ke depan. *Script* “node baru = new node(data)” merupakan pembuatan objek *node* baru. Pada *script* “if(head==null)” merupakan sebuah kondisi yang mana jika kondisi terpenuhi maka akan melakukan proses inisialisasi variabel “head=baru; ” dan “tail=baru; ”, namun jika kondisi tidak terpenuhi maka akan melakukan proses “else” yang di dalamnya terdapat inisialisasi variabel pula “baru.prev=tail;”, “tail.next=baru;” dan “tail=baru;”

|  |
| --- |
| void primTerr(no2 trr, no2 rawat){  node Terran = head;  while (Terran != null){  if ( prim(Terran.data) ){  rawat.addLast(Terran.data);  }  Terran = Terran.next;  }  }  static boolean prim(int d){  int sep = 0;  if ( d < 1 ){  return false;  }  for ( int i = 1; i<=d; i++ ){  if ( d%i == 0 ){  sep++;  }  }  if (sep == 2){  return true;  } else {  return false;  }  } |

*Code* “void primTerr(no2 trr, no2 rawat){” merupakan pembuatan sebuah *method* dengan nama “primTerr”. *Script* ini berfungsi untuk memasukkan pasukan-pasukan Terran yang terkena parasit ke dalam *list* pasukan yang mendapat perawatan dengan menggunakan bantuan *method* “static boolean(int d){” yang menentukan bilangan prima di antara para pasukan Terran. Dalam *code* “static boolean prim(int d){” ini menentukan bilangan prima yang terdapat dari hasil *output* bilangan *random.* Cara kerja dari *method* ini adalah memeriksa kondisi dari data apakah membentuk bilangan prima atau tidak. Dengan cara memeriksa setiap data, apabila data memiliki dua buah faktor yaitu dapat membagi dirinya sendiri, dan dibagi satu maka dapat dikatakan prima. Apabila data tidak memiliki dua faktor maka tidak dapat dikatakan prima.

|  |
| --- |
| void fiboPros(no2 protosz, no2 rawat){  node protos = head;  while (protos != null){  if ( fibona(protos.data) ){  rawat.addLast(protos.data);  }  protos = protos.next;  }  }  static boolean fibona(int d){  int st = 0, nd = 1, fib = 0;  while( fib < d ){  fib = st+nd;  st = nd;  nd = fib;  }  if ( d == fib ){  return true;  }  else {  return false;  }  } |

*Code* “void fiboPross(no2 protosz, no2 rawat){” merupakan pembuatan sebuah *method* dengan nama “primTerr”. *Code* ini berfungsi untuk memasukkan pasukan-pasukan Protoss yang terkena parasit ke dalam *list* pasukan yang mendapat perawatan dengan menggunakan bantuan *method* “static fibona(int d){” yang menentukan bilangan *Fibonacci* di antara para pasukan Protoss. Dalam *code* “static boolean fibona(int d){” ini menentukan bilangan prima yang terdapat dari hasil *output* bilangan *random.* Cara kerja dari *method* ini adalah memeriksa kondisi dari data apakah membentuk bilangan prima atau tidak. Dengan cara memeriksa setiap data, apabila data memiliki dua buah faktor yaitu dapat membagi dirinya sendiri, dan dibagi satu maka dapat dikatakan prima. Apabila data tidak memiliki dua faktor maka tidak dapat dikatakan prima.

|  |
| --- |
| void display(){  node cur = head;  while (cur != null){  System.out.print(cur.data+" ");  cur = cur.next;  }  System.out.println("\n");  } |

*Code* “public void tampil(){” merupakan pembuatan sebuah *method* bernama “tampil”. *Method* ini berfungsi untuk menampilkan nilai dari setiap *node* yang ada. *Script* “node tnjk;” merupakan pembuatan variabel bertipe data *node* lalu variabel tersebut diinisialisasikan dengan variabel “cur=head;”. *Method* ini digunakan untuk menampilkan keseluruhan pasukan tanpa diurutkan.

|  |
| --- |
| public static void main(String[] args) {  Random r = new Random();  no2 barisan = new no2();  no2 Terran = new no2();  no2 Protoss = new no2();  no2 perawatan = new no2(); |

*Code* “public static void main (String[] args)” merupakan *method* *main* dimana digunakan untuk menaruh semua perintah yang akan dijalankan oleh programdan dilakukan pendeklarasian *object* agar program dapat menjalankan *method*-*method* di kelas “no2”.

|  |
| --- |
| for(int i=0; i<30; i++){  int masuk = r.nextInt(100);  barisan.addLast(masuk);  if (masuk<=50){  Terran.addLast(masuk);  } else if (masuk>50 || masuk<=100){  Protoss.addLast(masuk);  } i++;  } |

*Code* “public static void main (String[] args)” merupakan *method* main, digunakan untuk menaruh semua perintah yang akan dijalankan oleh programdan dilakukan pendeklarasian *object* agar program dapat menjalankan *method*-*method* di kelas “no2”. *Code* “for(int i=0; i<30; i++){” dan *Code* “int masuk = r.nextInt(100);” digunakan untuk melakukan perulangan sebanyak 30 kali, yang dimana ada operasi *increment* dan mendeklarasikan variabel “masuk” untuk menyimpan “r” yang dijadikan sebagai *input* untuk menampilkan nilai random dengan *range* dari 1-100. Lalu dengan mengakses method “addLast(masuk)” yang dijadikan sebagai inputan untuk penambahan nilai pada data. Kemudian, diberikan kondisi jika variabel “masuk” lebih kecil sama dengan 50 maka memanggil objek “Terran” untuk mengakses *method* “addLast(masuk)”. Lalu, diberikan kondisi jika variabel “masuk” lebih besar dari 50 dan variabel “masuk” lebih kecil sama dengan 100 maka memanggil objek “protos” untuk mengakses *method* “addLast(masuk)” dan dilakukan operasi *increment* pada variabel “i”.

|  |
| --- |
| System.out.println("Barisan Pasukan: ");  barisan.display();  System.out.println("Barisan Terran: ");  Terran.display();  System.out.println("Barisan Protoss: ");  Protoss.display();  Terran.primTerr(Terran, perawatan);  System.out.println("Pasukan Terran yang terkena parasi (bilangan prima): ");  perawatan.display();  Protoss.fiboPros(Protoss, perawatan);  System.out.println("Pasukan yang terkena parasit setelah ditambah Pasukan Protoss (bilangan fibonaci): ");  perawatan.display();  } |

*Code* “barisan.display();” berfungsi untuk memanggil *method* *display* menggunakan *object* yang telah dibuat, ini agar data dalam *method* yang dipanggil tereksekusi dapat ditampilkan. Hal ini juga berlaku terhadap *syntax* yang menggunakan *method* yang sama.

* 1. **KESIMPULAN**

Berdasarkan praktikum modul 3 yang telah dilakukan, dapat disimpulkan bahwa:

1. *Double linked list sorting* merupakan cara untuk mengurutkan nilai–nilai yang ada pada *double linked list* ada banyak cara mengurutkan yaitu *bubble sort*, *selection sort*, dan *insertion sort*.
2. Mengimplementasikan *double linked list* untuk *searching* data menggunakan metode *sequential search* dan *binary search* untuk mempermudah mencari data yang diinginkan oleh *user*.