**UJIAN TENGAH SEMESTER**

**ANALISIS ALGORITMA**

****

**OLEH:**

**ASEP RIDWAN HIDAYAT**

**231012050036**

**01MKME001-REGULAR C**

**PROGRAM STUDI MAGISTER TEKNIK INFORMATIKA**

**PROGRAM PASCA SARJANA**

**UNIVERSITAS PAMULANG**

**TANGERANG SELATAN**

**2024**

**Jawaban No 1**

1. **PERHITUGAN ANALITIK**

|  |
| --- |
| Diketahui dari soal data sebagai berikut   * Memiliki 4 simpul (A, B, C, D) dan 5 resistor * *Rab=5Ω, Rac=3Ω, Rbc=6Ω, Rbd=8Ω, Rcd=4Ω*   Misalkan :  Diketahui arus yang mengalir melalui resistansi *RAB , Rac*, *Rbc, Rbd, Rcd*, berturut-turut sebagai *ZAB, ZAC, ZBC, ZBD, dan ZCD*.  gunakan hukum Kirchhoff kedua pada simpul A, B, C kita bisa menyusun persamaan sebagai berikut   1. **Simpul A**   *ZAB*= *ZAC*   1. **Simpul B** 2. **Simpul C**   Hukum Kirchhoff pertama untuk membentuk sebuah sistem persamaan linear. Misalnya, kita bisa menggunakan loop ABC:   1. **Loop ABC**   𝑉𝐴𝐵 = *V*𝐴C + *VBC*  (Substitusi nilai resistor)  Jadi ada 4 pesamaan dari atas:      Dari persamaan kedua (Simpul A) untuk menggantikan *ZAB* dalam persamaan 4 (Loop ABC), menjadi:  Ada dua persamaan dengan hanya dua variabel, yaitu dan yang dapat diselesaikan, seperti dibawah ini:  Dari perhitungan diatas didapat:   1. **(1)** 2. **(2)**   Subsitusi persamaan (1) ke persamaan (2) untuk menentukan nilai Z*BD ,* didapat  substitusi kembali nilai ke dalam persamaan (2) untuk menemukan nilai  Sekarang kita punya nilai  **.** dan karena **= 3** kita bisa menghitung  Jadi bisa didapatkan nilai yang lainnya seperti dibawah ini:  Jadi didapatkan nilai: |

1. **ALGORITMA**

|  |
| --- |
| def solve\_circuit():  # Resistansi antara simpul  R\_AB = 5  R\_AC = 3  R\_BC = 6  R\_BD = 8  R\_CD = 4  # Menghitung nilai arus  Z\_BC = 3 # Asumsikan nilai awal untuk Z\_BC  Z\_AC = 3 \* Z\_BC  Z\_AB = Z\_AC  Z\_BD = 6 \* Z\_BC  Z\_CD = Z\_AC - Z\_BC  # Menampilkan hasil  print("Arus yang mengalir melalui setiap resistor:")  print(f"Z\_AB = {Z\_AB} A")  print(f"Z\_AC = {Z\_AC} A")  print(f"Z\_BC = {Z\_BC} A")  print(f"Z\_BD = {Z\_BD} A")  print(f"Z\_CD = {Z\_CD} A")  if \_\_name\_\_ == "\_\_main\_\_":  solve\_circuit() |

1. **PROGRAM Python**

|  |
| --- |
| def solve\_circuit(R\_AB, R\_AC, R\_BC, R\_BD, R\_CD):  # Menghitung nilai arus  Z\_BC = 3 # Asumsikan nilai awal untuk Z\_BC  Z\_AC = 3 \* Z\_BC  Z\_AB = Z\_AC  Z\_BD = 6 \* Z\_BC  Z\_CD = Z\_AC - Z\_BC  # Menampilkan hasil  print("Arus yang mengalir melalui setiap resistor:")  print(f"Z\_AB = {Z\_AB} A")  print(f"Z\_AC = {Z\_AC} A")  print(f"Z\_BC = {Z\_BC} A")  print(f"Z\_BD = {Z\_BD} A")  print(f"Z\_CD = {Z\_CD} A")  if \_\_name\_\_ == "\_\_main\_\_":  # Input nilai resistansi antara simpul  R\_AB = float(input("Masukkan nilai resistansi antara simpul A dan B (ohm): "))  R\_AC = float(input("Masukkan nilai resistansi antara simpul A dan C (ohm): "))  R\_BC = float(input("Masukkan nilai resistansi antara simpul B dan C (ohm): "))  R\_BD = float(input("Masukkan nilai resistansi antara simpul B dan D (ohm): "))  R\_CD = float(input("Masukkan nilai resistansi antara simpul C dan D (ohm): "))  # Memanggil fungsi untuk menyelesaikan sirkuit  solve\_circuit(R\_AB, R\_AC, R\_BC, R\_BD, R\_CD) |

1. **OUTPUT PROGRAM**

|  |
| --- |
|  |

**Jawaban No 2**

1. **Buktikan 3n dan 5n selalu habis dibagi oleh 8**
2. **PERHITUNGAN SECARA ANALITIK**

|  |
| --- |
| Langkah induksi pertama :   1. Untuk n = 1   3n dan 5n selalu habis dibagi oleh 8  31 = 3 (**tidak habis dibagi 8**)  51 = 5 (**tidak habis dibagi 8**)  Ini artinya pernyataan diatas tidak benar untuk n=1   1. Untuk n = z (kita asumsikan pernyataan ini benar untuk n = z)   Asumsikan k bilangan bulangan positif dan 3k dan 5k habis dibagi 8, dengan kata lain dengan danbilangan bulat positif   1. Buktikan untuk n = z + 1   Jika asumsi n = z benar untuk pernyataan diatas (3n dan 5n selalu habis dibagi oleh 8), maka pernyataan tersebut juga benar untuk n = z + 1.  Mari kita coba:  Dari asumsi kita sebelumnya, diketahui bahwa habis dibagi 8 untuk n = z  Maka,jika ditulis ulang persamaan nya sebagai berikut:       Karena 24 dan 40 adalah kelipatan dari 8, maka juga habis dibagi 8  Dengan demikian, dengan metode induksi matematika, kita telah membuktikan bahwa untuk setiap bilangan bulat positif n, 3n dan 5n selalu habis dibagi oleh 8 dan n |

1. **ALGORITMA**

|  |
| --- |
| 1. Fungsi Induksi\_Matematika(n): //buat fungsi untuk input n 2. //jika n =1 tidak terpenuhi karena n 3. Jika n = 1: 4. Tampilkan "Pernyataan tidak berlaku untuk n = 1." 5. Berhenti. 6. Jika n > 1: 7. Untuk i dari 1 hingga n: 8. Jika (3^n) / 8 ≠ 0 atau (5^n) / 8 ≠ 0: 9. Tampilkan "Pernyataan tidak berlaku untuk n =", i. 10. Berhenti. 12. Tampilkan "Pernyataan benar untuk semua bilangan bulat positif hingga n." |

1. **PEMOGRAMAN Python**

|  |
| --- |
| def Induksi\_Matematika(n):      if n == 1:          print("Pernyataan tidak berlaku untuk n = 1.")          return      for i in range(1, n+1):          if (3\*\*n) % 8 != 0 or (5\*\*n) % 8 != 0:              print("Pernyataan tidak berlaku untuk n =",i)              return      else:          print("Pernyataan benar untuk semua bilangan bulat positif hingga",n)  def main():      try:          n = int(input("Masukkan nilai n: "))          if n <= 0:              print("Masukkan bilangan bulat positif yang lebih besar dari 0.")              return          Induksi\_Matematika(n)      except ValueError:          print("Masukkan bilangan bulat.")  if \_\_name\_\_ == "\_\_main\_\_":      main() |

1. **OUTPUT**

|  |
| --- |
|  |

**Jawaban No. 3**

1. **PERHITUNGAN ANALITIK**

Untuk memodifikasi algoritma Heap Sort agar dapat mengurutkan array dengan efisien dalam kondisi terbaik dan terburuk, diantaranya dengan langkah berikut:

1. **Kondisi terbaik** untuk algoritma Heap Sort terjadi ketika array masukan sudah dalam bentuk heap maksimum atau heap minimum (bergantung pada jenis Heap yang digunakan). Dalam hal ini, algoritma hanya membutuhkan waktu O(n) untuk membangun heap, karena tidak ada perubahan yang diperlukan untuk memastikan properti heap terpenuhi.
2. **Kondisi terburuk** untuk algoritma Heap Sort terjadi ketika elemen-elemen dalam larik berada dalam urutan terbalik dari urutan yang diinginkan. Dalam kondisi ini, setiap kali sebuah elemen dimasukkan ke dalam heap saat membangunnya, elemen tersebut harus bergeser ke atas melewati semua elemen yang ada dalam heap untuk mencapai posisinya yang seharusnya. Hal ini menyebabkan kompleksitas waktu yang paling buruk dalam setiap iterasi membangun heap.

Langkah-langkah optimisasi Heapify dengan Bottom-up Approach:

1. Ubah proses membangun heap dari pendekatan top-down menjadi bottom-up.
2. Lakukan heapify untuk setiap node dimulai dari node terakhir yang bukan daun hingga root
3. **ALGORITMA**

Algoritma Psudo code seperti ini untuk pengurutan heapify\_bottom\_up:

|  |
| --- |
| def heapify\_bottom\_up(arr, n, i):  largest = i  left = 2 \* i + 1  right = 2 \* i + 2  if left < n and arr[left] > arr[largest]:  largest = left  if right < n and arr[right] > arr[largest]:  largest = right  if largest != i:  arr[i], arr[largest] = arr[largest], arr[i]  heapify\_bottom\_up(arr, n, largest)  def heap\_sort\_bottom\_up(arr):  n = len(arr)  # Membangun heap secara bottom-up  for i in range(n // 2 - 1, -1, -1):  heapify\_bottom\_up(arr, n, i)  # Ekstraksi elemen satu per satu dari heap  for i in range(n-1, 0, -1):  arr[i], arr[0] = arr[0], arr[i]  heapify\_bottom\_up(arr, i, 0)  # Contoh penggunaan:  arr = [12, 11, 13, 5, 6, 7]  heap\_sort\_bottom\_up(arr)  print("Array yang diurutkan adalah:", arr) |

1. **PROGRAM Python**

|  |
| --- |
| def heapify\_bottom\_up(arr, n, i):  largest = i  left = 2 \* i + 1  right = 2 \* i + 2  if left < n and arr[left] > arr[largest]:  largest = left  if right < n and arr[right] > arr[largest]:  largest = right  if largest != i:  arr[i], arr[largest] = arr[largest], arr[i]  heapify\_bottom\_up(arr, n, largest)  def build\_max\_heap(arr):  n = len(arr)  # Starting from the last non-leaf node and heapify them in reverse order  for i in range(n // 2 - 1, -1, -1):  heapify\_bottom\_up(arr, n, i)  # Input larik dari pengguna  arr = list(map(int, input("Masukkan elemen: ").split()))  # Membangun heap maksimum  build\_max\_heap(arr)  print("Heap maksimum:", arr) |

1. **Output**

Implementasi dengan menggunakan python :

|  |
| --- |
|  |

**Jawaban No. 4**

1. **Perhitungan Analitik**

Untuk menghitung jumlah operasi pembandingan yang dilakukan dalam algoritma Merge Sort untuk mengurutkan array [5, 1, 6, 2, 3, 4, 7] secara menurun dengan aloritman Merge Sort (perbandingan antara elemen-elemen dalam subarray)

1. Saat membagi array menjadi subarray, array akan dibagi menjadi subarray berikut:

* [5, 1, 6, 2] dan [3, 4, 7]
* [5, 1] dan [6, 2]
* [5] dan [1]
* [6] dan [2]
* [3, 4] dan [7]
* [3] dan [4]

1. Selanjutnya, subarray akan digabungkan secara berurutan untuk menghasilkan array terurut:

* [1, 5, 2, 6] dan [3, 4, 7]
* [1, 5] dan [2, 6]
* [1] dan [5]
* [2] dan [6]
* [3, 4] dan [7]
* [3] dan [4]

dalam penggabungan array jumlah elemen dalam subarray adalah n/2, di mana n adalah panjang array. karena jumlah langkah yang diperlukan adalah 2 kali subarray.

maka:

Didalam array ini [5, 1, 6, 2, 3, 4, 7] Panjang array adalah 7 berarti jika disubstitusikan seperti ini operasi perbandingan.

1. **Algoritma**

Psudo dengan python

|  |
| --- |
| def merge\_sort(arr):  if len(arr) > 1:  mid = len(arr) // 2  left\_half = arr[:mid]  right\_half = arr[mid:]  merge\_sort(left\_half)  merge\_sort(right\_half)  i = j = k = 0  # Menggabungkan dua bagian menjadi arr dalam urutan menurun  while i < len(left\_half) and j < len(right\_half):  if left\_half[i] >= right\_half[j]:  arr[k] = left\_half[i]  i += 1  else:  arr[k] = right\_half[j]  j += 1  k += 1  # Menyalin elemen yang tersisa dari left\_half, jika ada  while i < len(left\_half):  arr[k] = left\_half[i]  i += 1  k += 1  # Menyalin elemen yang tersisa dari right\_half, jika ada  while j < len(right\_half):  arr[k] = right\_half[j]  j += 1  k += 1  # Masukan Array:  arr = [5, 1, 6, 2, 3, 4, 7]  merge\_sort(arr)  print("Array setelah diurutkan secara menurun:", arr) |

1. **Pemograman**

|  |
| --- |
| def merge\_sort(arr):      if len(arr) > 1:          mid = len(arr) // 2          left\_half = arr[:mid]          right\_half = arr[mid:]          merge\_sort(left\_half)          merge\_sort(right\_half)          i = j = k = 0          # Menggabungkan dua bagian menjadi arr dalam urutan menurun          while i < len(left\_half) and j < len(right\_half):              if left\_half[i] >= right\_half[j]:                  arr[k] = left\_half[i]                  i += 1              else:                  arr[k] = right\_half[j]                  j += 1              k += 1          # Menyalin elemen i yang tersisa dari right\_half, jika ada          while i < len(left\_half):              arr[k] = left\_half[i]              i += 1              k += 1          # CMenyalin elemen j yang tersisa dari right\_half, jika ada          while j < len(right\_half):              arr[k] = right\_half[j]              j += 1              k += 1  # array dari soal  arr = [5, 1, 6, 2, 3, 4, 7]  merge\_sort(arr)  print("Array setelah diurutkan secara menurun (desc):", arr) |

1. **Output**

Implementasi dengan menggunakan python :

|  |
| --- |
|  |