**Jawaban No 1**

1. **Perhitugan Analitik**

|  |
| --- |
| Diketahui   * Hukum Kirchhoff I (Hukum Kirchhoff Arus) digunakan untuk menyusun persamaan arus di setiap simpul. * Hukum Kirchhoff II (Hukum Kirchhoff Tegangan)digunakan untuk menyusun persamaan tegangan di setiap loop.   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   Mari kita asumsikan arah arus dari simpul A ke simpul B dan dari simpul B ke simpul C.   1. Terapkan hukum Kirchhoff I untuk simpul:  * Pada simpul A: Arus masuk = arus keluar, * Pada simpul B: Arus masuk = arus keluar * Pada simpul C: Arus masuk = arus keluar * Pada simpul D: Arus masuk = arus keluar  1. Terapkan hukum Kirchhoff II untuk loop:  * Pilih satu loop (misalnya, ABCA atau BCDB) dan tulis persamaan yang menjelaskan jumlah tegangan jatuh dan naik di sepanjang loop tersebut. Kemudian, gunakan hukum Ohm (V = IR) untuk mengekspresikan tegangan dalam istilah arus dan resistansi.   Kita Hitung arus yang mengalir melalui setiap resistor:   1. **Arus pada Resistor RAB (antara simpul A dan B):**   Kita asumsikan arus 𝐼𝐴𝐵 mengalir dari A ke B.  𝑉𝐴𝐵=𝐼𝐴𝐵×𝑅𝐴𝐵  𝑉𝐴𝐵 =IAB×5  ​   1. **Arus pada Resistor RAC (antara simpul A dan C)**   Kita asumsikan arus 𝐼𝐴𝐶 mengalir dari A ke C.  𝑉𝐴𝐶=𝐼𝐴𝐶×𝑅𝐴𝐶  𝑉𝐴𝐶=𝐼𝐴𝐶×   1. **Arus pada Resistor RBC (antara simpul B dan C):**   Kita asumsikan arus 𝐼𝐵𝐶​ mengalir dari B ke C.  𝑉𝐵𝐶=𝐼𝐵𝐶×𝑅𝐵𝐶  𝑉𝐵𝐶=𝐼𝐵𝐶×6   1. Arus pada Resistor RBD (antara simpul B dan D):   Kita asumsikan arus 𝐼𝐵𝐷​ mengalir dari B ke D.  𝑉𝐵𝐷=𝐼𝐵𝐷×𝑅𝐵𝐷  𝑉𝐵𝐷=𝐼𝐵𝐷×8   1. Arus pada Resistor 𝑅𝐶𝐷 (antara simpul C dan D):   Kita asumsikan arus 𝐼𝐶𝐷​ mengalir dari C ke D.  𝑉𝐶𝐷=𝐼𝐶𝐷×𝑅𝐶𝐷  𝑉𝐶𝐷 = 𝐼𝐶𝐷 ×4  Dalam loop ABCA, kita memiliki dua tegangan jatuh (VB - VA dan VC - VA) dan dua tegangan naik (VA - VB dan VA - VC).  Menurut hukum Kirchhoff untuk tegangan dalam loop tertutup, jumlah tegangan jatuh harus sama dengan jumlah tegangan naik.  Jadi, kita dapat menulis persamaan:  (VA−VB) + (VA−VC) = (VB−VA) + (VC−VA)  Kita telah mengekspresikan tegangan dalam istilah arus dan resistansi. Sekarang kita bisa menggunakan hubungan antara tegangan, arus, dan resistansi (V = IR) untuk mengekspresikan tegangan dalam istilah arus dan resistansi.  Dengan mengganti 𝑉 dengan 𝐼𝑅, kita bisa menulis:  (𝐼𝐴𝐵×𝑅𝐴𝐵)+(𝐼𝐴𝐶×𝑅𝐴𝐶)=(𝐼𝐵𝐶×𝑅𝐵𝐶)+(𝐼𝐶𝐷×𝑅𝐶𝐷)  Sekarang, kita memiliki satu persamaan dengan lima variabel arus yang tidak diketahui (𝐼𝐴𝐵​ , 𝐼𝐴𝐶​ , 𝐼𝐵𝐶​ , 𝐼𝐵𝐷​ , 𝐼𝐶𝐷 ).  Namun, kita juga memiliki keterkaitan arus di simpul yang dapat membantu kita memecahkan sistem persamaan ini.  maka didapat 4 persamaan dari hukum Kirchhoff untuk simpul dan Persamaan tegangan untuk loop ABCA   1. Di simpul A: 𝐼𝐴𝐵+𝐼𝐴𝐶=𝐼𝐵𝐶 2. Di simpul B: 𝐼𝐵𝐶=𝐼𝐵𝐷 3. Di simpul C: 𝐼𝐴𝐶=𝐼𝐶𝐷 4. (𝐼𝐴𝐵×𝑅𝐴𝐵)+(𝐼𝐴𝐶×𝑅𝐴𝐶)=(𝐼𝐵𝐶×𝑅𝐵𝐶)+(𝐼𝐶𝐷×𝑅𝐶𝐷)   Dari 4 persamaan ini bisa dilakukan perhitungan secara *eliminasi*, secara *distribusi* atau *matriks gaus Jordan* untuk mendapatkan nilai tiap resistor |

1. **Algoritma**

|  |
| --- |
| # Inisialisasi resistansi  RAB = 5  RAC = 3  RBC = 6  RBD = 8  RCD = 4  # Terapkan hukum Kirchhoff untuk simpul  # Persamaan untuk simpul A: IAB + IAC = IBC  # Persamaan untuk simpul B: IBC = IBD  # Persamaan untuk simpul C: IAC = ICD  # Terapkan hukum Kirchhoff untuk loop  # Persamaan untuk loop ABCA: 5IAB + 3(IAC - IAB) = 6IBC + 4ICD  # Bentuk sistem persamaan  # Kita memiliki persamaan di atas bersama dengan persamaan untuk simpul  # Gunakan metode substitusi atau eliminasi untuk menyelesaikan sistem persamaan  # Misalnya, di sini kita menggunakan metode eliminasi  from sympy import symbols, solve  IAB, IAC, IBC, IBD, ICD = symbols('IAB IAC IBC IBD ICD')  # Persamaan untuk simpul  eq1 = IAB + IAC - IBC  eq2 = IBC - IBD  eq3 = IAC - ICD  # Persamaan untuk loop  eq4 = 5 \* IAB + 3 \* (IAC - IAB) - 6 \* IBC - 4 \* ICD  # Selesaikan sistem persamaan  solution = solve((eq1, eq2, eq3, eq4), (IAB, IAC, IBC, ICD))  # Keluarkan hasil  print("Nilai arus yang mengalir melalui setiap resistor:")  print(f"IAB = {solution[IAB]}")  print(f"IAC = {solution[IAC]}")  print(f"IBC = {solution[IBC]}")  print(f"IBD = {solution[IBC]}") # Karena IBC = IBD  print(f"ICD = {solution[ICD]}") |

**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 = k (kita asumsikan pernyataan ini benar untuk n=k)   Asumsikan k bilangan bulangan positif dan 3k dan 5k habis dibagi 8, dengan kata lain dengan danbilangan bulat positif   1. Buktikan untuk n = k+1   Jika asumsi n =k benar untuk pernyataan diatas (3n dan 5n selalu habis dibagi oleh 8), maka pernyataan tersebut juga benar untuk n = k + 1.  Mari kita coba:  Dari asumsi kita sebelumnya, diketahui bahwa habis dibagi 8 untuk n=k  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**

Menggunakan 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() |

**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) |

**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) |

Output

Implementasi dengan menggunakan python :

|  |
| --- |
|  |