Nama : Moh. Syahrul Aziz Imastara

Nim : 1203220084

Kelas : IF-02-01

1. Rekursif

```c
#include <stdio.h>
unsigned long long factorial(int n) {
 // Base case: faktorial dari 0 dan 1 adalah 1
  if (n == 0 || n == 1) {
  return 1;
        }
   // Recursive case: faktorial dari n adalah n * faktorial(n - 1)
  else {
     return n * factorial(n - 1);
  }
}
int main() {
  int num;
    printf("Masukkan bilangan untuk menghitung faktorial: ");
    scanf("%d", &num);
     // Memanggil fungsi faktorial
      unsigned long long result = factorial(num);
     // Menampilkan hasil
    printf("Faktorial dari %d adalah: %llu\n", num, result);
  return 0;
}
```

Kompleksitas waktu dari fungsi faktorial dalam kode di atas adalah O(n), di mana n adalah bilangan yang dimasukkan untuk dihitung faktorialnya. Ini karena setiap kali fungsi `factorial` dipanggil dengan nilai n, fungsi tersebut memanggil dirinya sendiri dengan nilai n – 1.

Kenapa kompleksitasnya adalah O(n):

1. Setiap pemanggilan rekursif dari fungsi faktorial mengurangi nilai n sebesar 1.
2. Dengan setiap pemanggilan rekursif, nilai n harus berkurang menuju base case.
3. Oleh karena itu, jumlah pemanggilan rekursif adalah n - 1, di mana n adalah bilangan asli yang diberikan.
4. Ini menciptakan rantai panggilan rekursif yang linier, yang menghasilkan kompleksitas waktu O(n).

2. Binary Search

```c
#include <stdio.h>
  // Fungsi rekursif untuk pencarian biner
int binarySearch(int arr[], int low, int high, int key) {
  // Base case: jika low lebih besar dari high, artinya elemen tidak ditemukan
  if (low > high)
  return -1;
  // Mencari nilai tengah
  int mid = low + (high - low) / 2;
  // Jika elemen tengah adalah kunci, maka kembalikan indeksnya
  if (arr[mid] == key)
  return mid;
  // Jika elemen tengah lebih kecil dari kunci, cari di bagian kanan array
  else if (arr[mid] < key)
  return binarySearch(arr, mid + 1, high, key);
  // Jika elemen tengah lebih besar dari kunci, cari di bagian kiri array
  else
  return binarySearch(arr, low, mid - 1, key);
}
int main() {
  int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
  int n = sizeof(arr) / sizeof(arr[0]);
  int key = 14;
  int result = binarySearch(arr, 0, n - 1, key);
  if (result == -1)
  printf("Elemen tidak ditemukan\n");
  else
  printf("Elemen ditemukan pada indeks %d\n", result);
  return 0;
}
```

Kompleksitas waktu dari algoritma binary search dalam kode di atas adalah O(log n), di mana n adalah jumlah elemen dalam array yang diurutkan. Ini karena dalam setiap iterasi, ukuran rentang pencarian di kurangi menjadi setengah.

Alasan mengapa kompleksitasnya adalah O(log n):

1. Pada setiap iterasi, rentang pencarian dari low ke high dibagi dua.
2. Dengan setiap iterasi, ukuran rentang tersebut berkurang setengah dari ukuran sebelumnya.
3. Oleh karena itu, jumlah iterasi yang dibutuhkan untuk mencari elemen dalam rentang tersebut adalah logaritma basis 2 dari ukuran rentang.
4. Karena itu, kompleksitas waktu algoritma pencarian biner adalah O(log n).

ABSENSI KULTAM

ABSENSI TEAMS