**Eratosthenes Eleği**

#include <stdio.h>

#include <stdbool.h>

#define MAX 1000

void sieveOfEratosthenes(int n) {

bool isPrime[MAX + 1];

for (int i = 2; i <= n; i++) {

isPrime[i] = true;

}

for (int p = 2; p \* p <= n; p++) {

if (isPrime[p]) {

for (int i = p \* p; i <= n; i += p) {

isPrime[i] = false;

}

}

}

printf("Asal Sayılar: ");

for (int i = 2; i <= n; i++) {

if (isPrime[i]) {

printf("%d ", i);

}

}

printf("\n");

}

int main() {

int n;

printf("Bir üst sınırı girin: ");

scanf("%d", &n);

sieveOfEratosthenes(n);

return 0;

}

**Sezgisel Yöntemle**

#include <stdio.h>

#include <stdbool.h>

#include <time.h>

bool isPrime(int n) {

if (n <= 1) {

return false;

}

if (n <= 3) {

return true;

}

if (n % 2 == 0 || n % 3 == 0) {

return false;

}

for (int i = 5; i \* i <= n; i += 6) {

if (n % i == 0 || n % (i + 2) == 0) {

return false;

}

}

return true;

}

int main() {

int n;

printf("Bir üst sınırı girin: ");

scanf("%d", &n);

clock\_t start\_time = clock();

printf("Asal Sayılar: ");

for (int i = 2; i <= n; i++) {

if (isPrime(i)) {

printf("%d ", i);

}

}

printf("\n");

clock\_t end\_time = clock();

double elapsed\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

printf("Çalışma Süresi: %f s\n", elapsed\_time);

return 0;

}

**Araya Yerleştirme**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Rastgele sayı üreten fonksiyon

int randomInt(int min, int max) {

return min + rand() % (max - min + 1);

}

int main() {

// Dizi boyutu

int n = 1000;

// A, B ve C dizileri

int A[n], B[n], C[n];

// A dizisi sıralı şekilde dolduruluyor

for (int i = 0; i < n; i++) {

A[i] = i + 1;

}

// B dizisi tersten sıralı şekilde dolduruluyor

for (int i = 0; i < n; i++) {

B[i] = n - i;

}

// Rastgele sayılarla C dizisi dolduruluyor

srand(time(NULL));

for (int i = 0; i < n; i++) {

C[i] = randomInt(1, 1000);

}

// Dizilerin içeriğini yazdırma (örneğin ilk 10 eleman)

printf("A dizisi: ");

for (int i = 0; i < 10; i++) {

printf("%d ", A[i]);

}

printf("\n");

printf("B dizisi: ");

for (int i = 0; i < 10; i++) {

printf("%d ", B[i]);

}

printf("\n");

printf("C dizisi: ");

for (int i = 0; i < 10; i++) {

printf("%d ", C[i]);

}

printf("\n");

return 0;

}

**Kabarcık Sıralama**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int randomInt(int min, int max) {

return min + rand() % (max - min + 1);

}

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int n = 1000;

int A[n], B[n], C[n];

// A dizisi sıralı şekilde dolduruluyor

for (int i = 0; i < n; i++) {

A[i] = i + 1;

}

// B dizisi tersten sıralı şekilde dolduruluyor

for (int i = 0; i < n; i++) {

B[i] = n - i;

}

// Rastgele sayılarla C dizisi dolduruluyor

srand(time(NULL));

for (int i = 0; i < n; i++) {

C[i] = randomInt(1, 1000);

}

// A dizisini kabarcık sıralama ile sırala

bubbleSort(A, n);

// B dizisini kabarcık sıralama ile sırala

bubbleSort(B, n);

// Dizilerin içeriğini yazdırma (örneğin ilk 10 eleman)

printf("Sıralanmış A dizisi: ");

for (int i = 0; i < 10; i++) {

printf("%d ", A[i]);

}

printf("\n");

printf("Sıralanmış B dizisi: ");

for (int i = 0; i < 10; i++) {

printf("%d ", B[i]);

}

printf("\n");

printf("C dizisi: ");

for (int i = 0; i < 10; i++) {

printf("%d ", C[i]);

}

printf("\n");

return 0;

}

**Araya Yerleştirme Sıralama (Insertion Sort):**

İlk olarak, bir dizi elemanının başlangıcında bir eleman sıralı bir alt dizi olarak kabul edilir.

Ardından, sıralanmamış elemanlar sırayla alınır ve sıralı alt dizi içine yerleştirilir.

Dizi küçükse ve elemanlar nispeten sıralıysa etkili bir seçenektir.

En iyi durumda (dizi zaten sıralı) O(n) zaman karmaşıklığına sahiptir, ancak en kötü durumda (dizi tamamen ters sıralı) O(n^2) zaman karmaşıklığına sahiptir.

**Kabarcık Sıralama (Bubble Sort):**

Her iki eleman karşılaştırılır ve gerektiğinde yer değiştirilir, böylece büyük elemanlar dizinin sonuna doğru ilerler.

Sıralı olmayan elemanlar yukarıdan aşağıya doğru "kabarcıklar" gibi yükselir.

Performansı genellikle diğer sıralama algoritmalarına göre daha düşüktür. En kötü durumda ve ortalama durumda O(n^2) zaman karmaşıklığına sahiptir.

**Birleştirme Sıralama (Merge Sort):**

Birleştirme sıralama, diziyi ikiye böler, her iki alt diziyi sıralar ve ardından bu iki sıralı alt diziyi birleştirir.

Dizi bölünmeye devam edilir ve alt diziler sıralandıktan sonra birleştirilir.

Ortalama ve en kötü durumda O(n log n) zaman karmaşıklığına sahiptir, bu nedenle büyük diziler için daha etkilidir.

Ekstra bellek kullanımı gerektirebilir çünkü alt dizilerin birleştirilmesi sırasında ekstra bellek alanına ihtiyaç duyar.

Sonuç olarak, araya yerleştirme sıralama, küçük veya nispeten sıralı dizileri sıralamak için uygun bir seçenektir, ancak büyük dizilerde yavaş çalışabilir. Kabarcık sıralama, pratikte pek kullanılmayan bir algoritma olup, performansı diğer sıralama algoritmalarına göre düşüktür. Birleştirme sıralama, büyük dizileri sıralamak için etkili bir seçenektir ve genellikle hızlı ve istikrarlı bir sıralama algoritması olarak tercih edilir.

**Binary Search Algoritması**

#include <stdio.h>

// Özyinelemeli ikili arama işlevi

int binarySearch(int arr[], int left, int right, int target) {

if (right >= left) {

int mid = left + (right - left) / 2;

// Orta eleman hedefe eşitse, hedefi bulduk

if (arr[mid] == target) {

return mid;

}

// Orta eleman hedeften büyükse, sol yarıyı ara

if (arr[mid] > target) {

return binarySearch(arr, left, mid - 1, target);

}

// Orta eleman hedeften küçükse, sağ yarıyı ara

return binarySearch(arr, mid + 1, right, target);

}

// Hedef dizi içinde bulunamazsa -1 döndür

return -1;

}

int main() {

int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

int n = sizeof(arr) / sizeof(arr[0]);

int target = 12;

int result = binarySearch(arr, 0, n - 1, target);

if (result != -1) {

printf("Hedef %d, dizinin %d. indekste bulundu.\n", target, result);

} else {

printf("Hedef dizide bulunamadı.\n");

}

return 0;

}

**Strassen İki Matrisin Çarpımı**

#include <stdio.h>

// Matrisleri yazdırmak için yardımcı bir işlev

void printMatrix(int n, int A[][n]) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

printf("%d ", A[i][j]);

}

printf("\n");

}

}

// Matris toplama işlemi

void addMatrices(int n, int C[][n], int A[][n], int B[][n]) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

C[i][j] = A[i][j] + B[i][j];

}

}

}

// Matris çıkarma işlemi

void subtractMatrices(int n, int C[][n], int A[][n], int B[][n]) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

C[i][j] = A[i][j] - B[i][j];

}

}

}

// Matris çarpımı işlemi (Strassen Algoritması)

void strassen(int n, int C[][n], int A[][n], int B[][n]) {

if (n == 1) {

C[0][0] = A[0][0] \* B[0][0];

return;

}

int newSize = n / 2;

int A11[newSize][newSize], A12[newSize][newSize], A21[newSize][newSize], A22[newSize][newSize];

int B11[newSize][newSize], B12[newSize][newSize], B21[newSize][newSize], B22[newSize][newSize];

int C11[newSize][newSize], C12[newSize][newSize], C21[newSize][newSize], C22[newSize][newSize];

int M1[newSize][newSize], M2[newSize][newSize], M3[newSize][newSize], M4[newSize][newSize];

int M5[newSize][newSize], M6[newSize][newSize], M7[newSize][newSize];

int temp1[newSize][newSize], temp2[newSize][newSize];

// A ve B matrislerini dörde böler

for (int i = 0; i < newSize; i++) {

for (int j = 0; j < newSize; j++) {

A11[i][j] = A[i][j];

A12[i][j] = A[i][j + newSize];

A21[i][j] = A[i + newSize][j];

A22[i][j] = A[i + newSize][j + newSize];

B11[i][j] = B[i][j];

B12[i][j] = B[i][j + newSize];

B21[i][j] = B[i + newSize][j];

B22[i][j] = B[i + newSize][j + newSize];

}

}

// Yedi farklı matris çarpımını hesapla (M1, M2, ..., M7)

addMatrices(newSize, temp1, A11, A22);

addMatrices(newSize, temp2, B11, B22);

strassen(newSize, M1, temp1, temp2);

addMatrices(newSize, temp1, A21, A22);

strassen(newSize, M2, temp1, B11);

subtractMatrices(newSize, temp1, B12, B22);

strassen(newSize, M3, A11, temp1);

subtractMatrices(newSize, temp1, B21, B11);

strassen(newSize, M4, A22, temp1);

addMatrices(newSize, temp1, A11, A12);

strassen(newSize, M5, temp1, B22);

subtractMatrices(newSize, temp1, A21, A11);

addMatrices(newSize, temp2, B11, B12);

strassen(newSize, M6, temp1, temp2);

subtractMatrices(newSize, temp1, A12, A22);

addMatrices(newSize, temp2, B21, B22);

strassen(newSize, M7, temp1, temp2);

// C matrisinin parçalarını hesapla

addMatrices(newSize, temp1, M1, M4);

subtractMatrices(newSize, temp2, M7, M5);

addMatrices(newSize, C11, temp1, temp2);

addMatrices(newSize, C12, M3, M5);

addMatrices(newSize, C21, M2, M4);

subtractMatrices(newSize, temp1, M1, M2);

addMatrices(newSize, temp2, M3, M6);

addMatrices(newSize, C22, temp1, temp2);

// C matrisini birleştir

for (int i = 0; i < newSize; i++) {

for (int j = 0; j < newSize; j++) {

C[i][j] = C11[i][j];

C[i][j + newSize] = C12[i][j];

C[i + newSize][j] = C21[i][j];

C[i + newSize][j + newSize] = C22[i][j];

}

}

}

int main() {

int n = 4; // Matris boyutu (4x4 örnek olarak)

int A[n][n], B[n][n], C[n][n];

// Matris A ve B'yi doldur

printf("Matris A:\n");

printMatrix(n, A);

printf("\nMatris B:\n");

printMatrix(n, B);

// Strassen Algoritması ile matris çarpımını hesapla

strassen(n, C, A, B);

printf("\nMatris C (A \* B):\n");

printMatrix(n, C);

return 0;

}