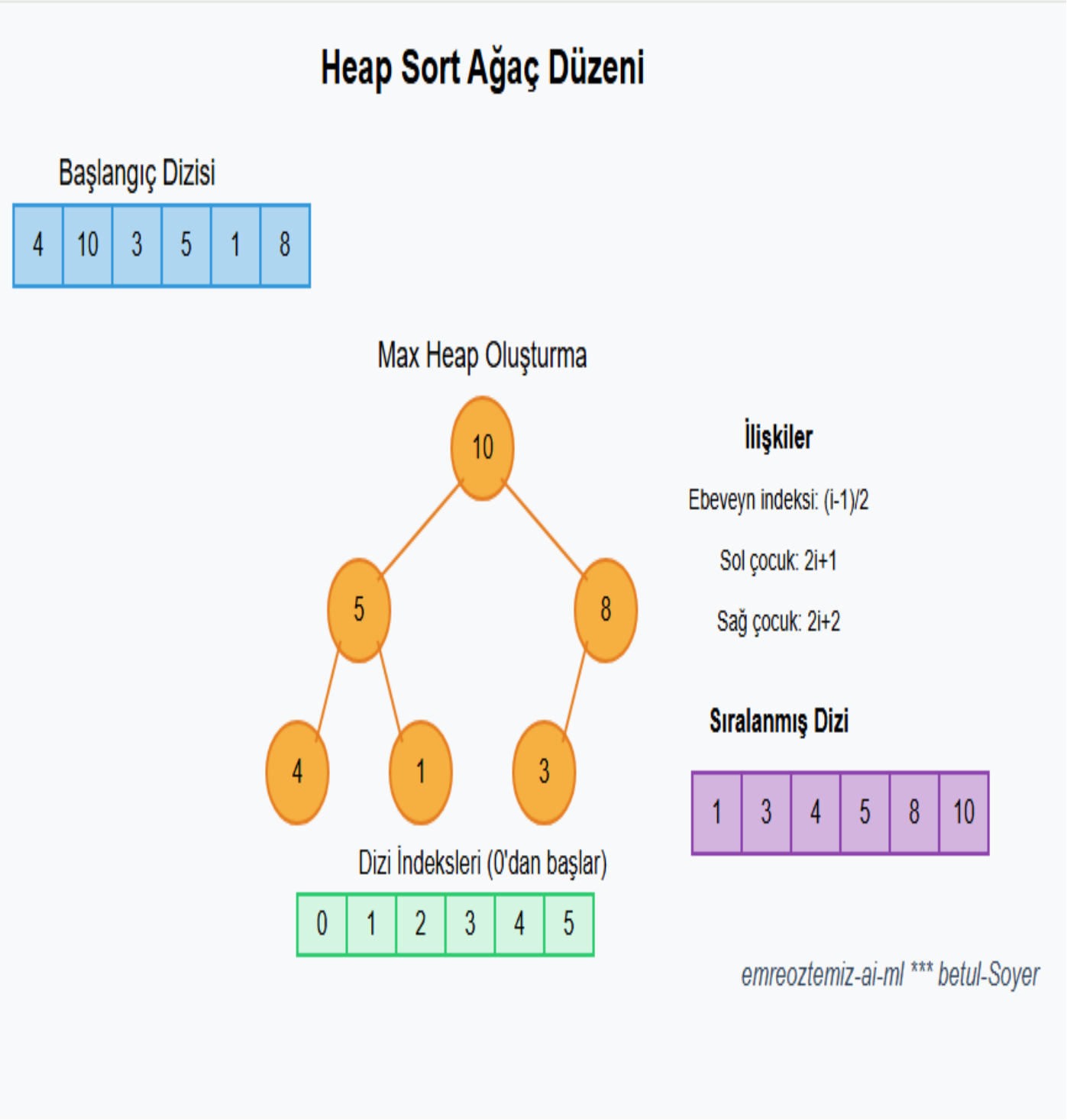
Heap Sort Algoritması



Giriş

HeapSort, karşılaştırma tabanlı bir sıralama algoritmasıdır ve heap (yığın) veri yapısını kullanır. İlk olarak 1964 yılında J.W.J. Williams tarafından geliştirilmiştir. HeapSort algoritması, sıralama problemlerine etkili bir çözüm sunar ve zaman karmaşıklığı açısından verimlidir. Bu doküman, HeapSort algoritmasının teorisini, uygulamasını ve performans özelliklerini detaylı şekilde inceleyecektir.

📚 Genel Bakış

Heap Sort, karşılaştırmaya dayalı bir sıralama algoritmasıdır ve binary heap veri yapısını kullanır. Özellikle büyük veri kümelerinde etkilidir.

Algoritma Yapısı 📚

- **Veri Yapısı:** Binary Heap (Max-Heap veya Min-Heap)
- **Tür:** Karşılaştırmalı Sıralamadır.
- **Kategori:** Yerinde Sıralama yapar. (In-place)
-

Zaman Karmaşıklığı 📚

Durum	Karmaşıklık
En Kötü Durum	O(n log n)
Ortalama Durum	O(n log n)
En İyi Durum	O(n log n)

Uzay Karmaşıklığı 📚

- **O(1)** Introduction to Algorithms isimli kitapta 396. Sf. da Zamanda ve mekanda önemli önemli asimptotik tasarruf sağladığından bahsetmektedir. Asimptotik verimlilik, bir algoritmanın girdi boyutu sonsuza doğru büyüdükçe nasıl performans gösterdiğini ifade eder. Heap Sort, yerinde (in-place) çalışır ve sabit miktarda ekstra bellek kullanır. Bu özellik, büyük veri setleriyle çalışırken önemli bir avantaj sağlar. İn-place çalışması sebebi ile yerinde sıralama yapar.
-

Temel Özellikler 📚

- **Kararlı Değil** (Stable sort değil)
- **Divide and Conquer** yaklaşımı kullanır.
- **Recursive** veya **iterative** olarak implemente edilebilir.

Çalışma Prensibi ⚙️

1. **Max-Heap Oluşturma:** Dizi max-heap yapısına dönüştürülür.
2. **Sıralama:** Kök eleman (en büyük) sürekli olarak dizinin sonuna yerleştirilir ve heap yeniden düzenlenir.

🌟 Kullanım Örnekleri

1. **Veritabanı Sistemleri**: Büyük veri setlerini sıralamak için kullanılır.
2. **İşletim Sistemleri**: Süreç önceliklerini yönetmek için.
3. **Grafik Algoritmaları**: Dijkstra ve Prim algoritmaları gibi öncelik kuyruğu gerektiren grafik algoritmalarında.
4. **Dış Sıralama (External Sorting)**: Bellek kapasitesinden daha büyük dosyaları sıralamak için.
5. **Medyan ve k. en büyük eleman bulma**: Heap yapısı kullanılarak verimli şekilde gerçekleştirilebilir.

✅ Avantajlar ve Dezavantajlar ❌

Avantajlar ✔

- **Tutarlı Performans**: Her durumda O(n log n) zaman karmaşıklığı sunar.
- **Bellek Verimliliği**: Yerinde (in-place) çalışır, ek bellek alanı gerektirmez.
- **Öncelik Kuyruğu Entegrasyonu**: Heap veri yapısı, öncelik kuyrukları için doğal bir temel sağlar.
- **Büyük Veri Setleri**: Bellek kısıtlaması olan ortamlarda büyük veri setlerini sıralamak için uygundur.

Dezavantajlar !!

- ***Algorithms*, Fourth Edition Kitap sf.342, PDF sf.355**: Kullanım yönü ile diğer sıralamalara göre kullanım alanı özellikle aynı değere sahip elemanların sırasını korumaz, yani kararlı değil (Unstable).
- **Pratik Uygulamalarda Yavaşlık**: Genellikle Quick Sort ve Merge Sort gibi diğer O(n log n) algoritmalardan daha yavaş çalışır, çünkü sabit faktörleri daha yüksektir.
- **Önbellek Kullanımı**: Heap yapısı, rastgele erişim modeliyle çalıştığından, modern CPU önbelleklerinde verimsiz olabilir.
- **Uygulaması Karmaşık**: Quicksort gibi bazı alternatiflere göre uygulaması daha karmaşıktır.

Performans Karşılaştırması 👀

Algoritma	En Kötü Durum	En İyi Durum	Yer karmaşıklığı
Heap Sort	O(nlogn)	O(nlogn)	O(1)
Quick Sort	$O(n^2)$	O(nlogn)	O(log n)
Merge Sort	O(nlogn)	O(nlogn)	O(n)

Proje Yapısı

Bu GitHub projesi aşağıdaki dosya ve klasörlerden oluşmaktadır:

```
heapsortprojesi/
├── Git Cmd screenshots/
│   ├── 1.PNG
│   ├── 10.PNG
│   ├── 11.PNG
│   ├── 12.PNG
│   ├── 13.PNG
│   ├── 14.PNG
│   ├── 15.PNG
│   ├── 16.PNG
│   ├── 17.PNG
│   ├── 18.PNG
│   ├── 19.PNG
│   ├── 2.PNG
│   ├── 3.PNG
│   ├── 4.PNG
│   ├── 5.PNG
│   ├── 6.PNG
│   ├── 7.PNG
│   ├── 8.PNG
│   ├── 9.PNG
│   └── Readme.md
├── heapsort-team.svg
├── odev.cpp
├── proje-tanitimi/
│   ├── finalproje.apk
│   └── Readme.md
├── README.md
├── TEST KODU/
│   ├── HeapSort-Test.cpp
│   └── Readme.md
└── TR-degisken-kullanimi/
    ├── Readme.md
    └── turkcedegiskenileheapsort.cpp
```

Örnek Kod (C++)💻

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Heap oluşturmak için yardımcı fonksiyonlarımız
// (arr:Düzenlenecek dizimiz, n:Dizinin boyutu, i:Kök düğüm indeksi)
void heapify(vector<int>& arr, int n, int i) {
    int largest = i; // En büyük elemanı kök olarak ayarlarız
    int left = 2 * i + 1; // Sol çocuk indeksimiz
    int right = 2 * i + 2; // Sağ çocuk indeksimiz

    // Sol çocuk kökten büyük ise
    if (left < n && arr[left] > arr[largest])
        largest = left;

    // Sağ çocuk şu anki en büyükten büyük ise
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // En büyük eleman kök değil ise
    if (largest != i) {
        swap(arr[i], arr[largest]); // Elemanları değiştirip

        // Etkilenen alt ağacı tekrar düzenleriz
        heapify(arr, n, largest);
    }
}

// Heap Sort ana fonksiyonumuz
void heapSort(vector<int>& arr) {
    int n = arr.size();

    // Max heap oluştur (diziyi heap yapısına çevirir)
    // Son yaprak olmayan düğümden başlayarak geriye gideriz
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Heap'ten elemanları tek tek çıkarırız
    for (int i = n - 1; i > 0; i--) {
        // Kök (en büyük eleman) ile son elemanı değiştiririz
        swap(arr[0], arr[i]);

        // Azaltılmış heap'i tekrar düzenleriz
        heapify(arr, i, 0);
    }
}

int main() {
    vector<int> arr;
    int n, num;

    // Kullanıcıdan eleman sayısını alırız
    cout << "Kac eleman gireceksiniz? ";
    cin >> n;

    // Kullanıcıdan elemanları alırız
    cout << n << " tane sayi giriniz:\n";
    for (int i = 0; i < n; i++) {
        cin >> num;
        arr.push_back(num);
    }
```

```
    // Sıralama öncesi diziyi gösteririz
    cout << "\nSiralama oncesi dizi: ";
    for (int num : arr) {
        cout << num << " ";
    }

    // Heap Sort uygulaması
    heapSort(arr);

    // Sıralama sonrası diziyi gösterir
    cout << "\nSiralama sonrasi dizi: ";
    for (int num : arr) {
        cout << num << " ";
    }

    return 0;
}
```

Örnek Kod Çıktısı ▣

- Kac eleman gireceksiniz? 5
- 5 tane sayi giriniz:
- 12 5 8 3 10
- Siralama oncesi dizi: 12 5 8 3 10
- Siralama sonrası dizi: 3 5 8 10 12

TEST KODU ▣

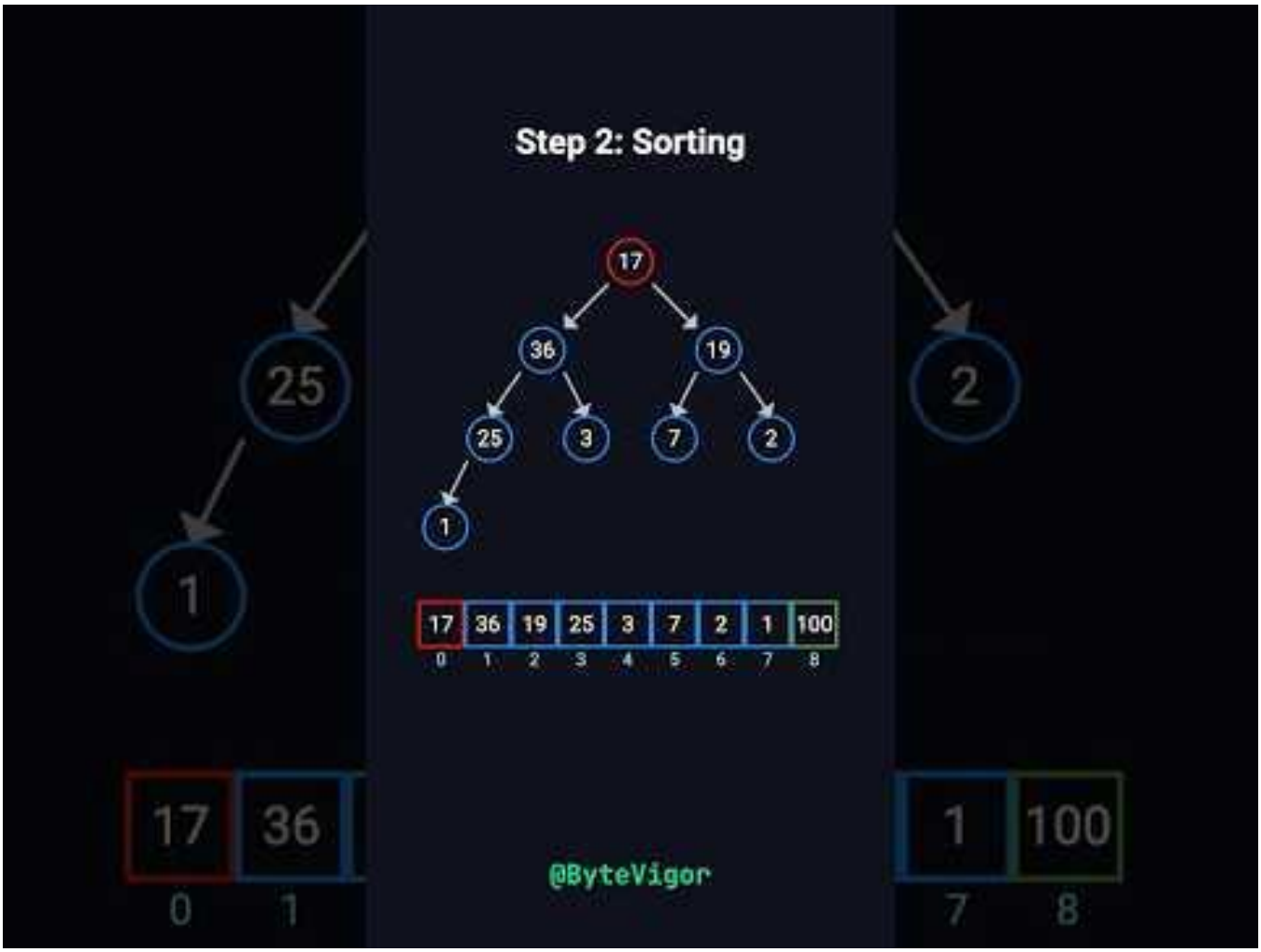
```
#include <iostream>
#include <vector>
using namespace std;

void heapify(vector<int>& arr, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(vector<int>& arr) {
    int n = arr.size();
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
```

```
        heapify(arr, i, 0);
    }
}

int main() {
    vector<int> arr = {5, 2, 8, 1, 9};

    cout << "Siralama oncesi dizi: ";
    for (int num : arr)
        cout << num << " ";

    heapSort(arr);

    cout << "\nSiralama sonrasi dizi: ";
    for (int num : arr)
        cout << num << " ";

    return 0;
}
```

TEST İÇİN CANLI SERVER

🔗 **Uygulamayı canlı hali ile test etmek için test kodunu, siteye girdikten sonraki ilgili alana yapıştırınız:**
👉 cpp-web-deneme.onrender.com Server Kurulum

Animasyon ◎



Kaynaklar

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Third Edition. MIT Press, 2009.
- Robert Sedgewick and Kevin Wayne. *Algorithms*, Fourth Edition. Addison-Wesley Professional, 2011.
- Heap Sort - GeeksforGeeks
- Binary Heap - Wikipedia
- Sadi Evren Şeker YT
- Markdown Kullanımı