**KARGO YÖNETİM SİSTEMİ**  
**Raporlama ve Performans Analizi**

**Ders Yetkilisi:**  
Serpil ÜSTEBAY

**Projeye Katkıda Bulunanlar:**

* Ali Emre Aydın (23120205056)
* Emir Başak Sunar (23120205053)
* Saruhan Görkem Türköz (23120205062)

**Ders:**  
Bilgisayar Mühendisliği, 2. Sınıf, Veri Yapıları ve Algoritmalar Proje Ödevi

NOT: Bu projede Pyside kütüphanesi kullanılmıştır. Kullanmaya başlamadan önce pyside kurmanız gerekmektedir, bunun için terminale:

pip install PySide6

yazmanız gerekmektedir. Sorun yaşıyorsanız aşağıdaki bağlantıyı ziyaret edin:  
https://pyside.readthedocs.io/en/latest/installing/windows.html

A computer screen shot of a computer

Description automatically generatedA computer screen shot of a computer

Description automatically generated

## 1. Bu Veri Yapıları Neden Seçildi?

1. **Bağlı Liste (Linked List):**
   * **Neden?** Müşterilerin gönderim geçmişini saklamak için kullanıldı. Bağlı liste, sıralı veri ekleme ve silme işlemleri için oldukça uygundur. Verilerin zaman sırasına göre düzenlenmesi gerektiği için bu yapı seçildi.
2. **Öncelik Kuyruğu (Priority Queue):**
   * **Neden?** Kargoların teslimat önceliğini belirlemek için kullanıldı. Bu yapı, daha hızlı teslim edilmesi gereken kargoların öncelikli olarak işleme alınmasını sağlar.
3. **Ağaç (Tree):**
   * **Neden?** Kargo rotalarını temsil etmek için kullanıldı. Ağaç yapısı, şehirler arası bağlantıları düzenler ve en kısa teslimat süresini hesaplamayı kolaylaştırır.
4. **Yığın (Stack):**
   * **Neden?** Müşterilerin son 5 kargosunu hızlıca göstermek için kullanıldı. Yığınlar, "Son Giren İlk Çıkar" (LIFO) mantığıyla çalışır ve bu tür işlemler için idealdir.
5. **Sıralama ve Arama Algoritmaları:**
   * **Neden?** Teslim edilmiş kargolar için **binary search**, teslim edilmemiş kargolar için ise **merge sort**algoritmaları kullanıldı. Binary search sıralı listelerde hızlı arama yaparken, merge sort büyük veri setlerini kararlı ve etkili şekilde sıralar.

## 2. Algoritmaların Zaman ve Bellek Karmaşıklıkları

| **Veri Yapısı/Algoritma** | **Zaman Karmaşıklığı** | **Bellek Karmaşıklığı** |
| --- | --- | --- |
| Linked List (Bağlı Liste) | O(n) (Ekleme/Silme) | O(n) |
| Priority Queue (Öncelik Kuyruğu) | O(log n) (Ekleme/Çıkarma) | O(n) |
| Binary Search (İkili Arama) | O(log n) | O(1) |
| Merge Sort (Birleştirme Sıralaması) | O(n log n) | O(n) |
| Tree (Ağaç Derinliği Hesaplama) | O(n) | O(h) (h: ağacın yüksekliği) |
| Stack (Yığın) | O(1) (Push/Pop) | O(n) |

### Önemli Noktalar:

* **Linked List:** İşlem süresi eleman sayısına bağlıdır.
* **Priority Queue:** Her ekleme veya çıkarma işlemi logaritmik maliyettedir.
* **Binary Search:** Sıralı listelerde oldukça hızlıdır.
* **Merge Sort:** Büyük veri kümelerinde etkili ve kararlı bir sıralama algoritmasıdır.
* **Tree:** Ağaç derinliği hesaplaması doğrusal zamanda yapılabilir.
* **Stack:** Push ve pop işlemleri sabit zamanda gerçekleşir.

A screenshot of a computer

Description automatically generated

## 3. Daha Verimli Çözümler Önerilebilir mi?

1. **Bağlı Liste Yerine HashMap:**
   * Müşteri bilgilerine daha hızlı erişmek için HashMap kullanılabilir. HashMap, O(1) erişim süresi sunar, ancak sıralı veri yönetimi için ek bir yapı gerekebilir.
2. **Merge Sort Yerine Quick Sort:**
   * Veriler rastgele sıralıysa, Quick Sort daha az bellek kullanabilir (O(log n) ek bellek). Ancak Merge Sort kararlılık açısından daha avantajlıdır.
3. **Priority Queue Yerine Dengeli İkili Arama Ağacı (BBST):**
   * Örneğin AVL veya Red-Black Tree gibi yapılar kullanılabilir. Bu yapılar hem sıralama hem de arama işlemlerinde O(log n) karmaşıklık sunar.
4. **Ağaç Yerine Graf Yapısı:**
   * Şehirler arası rotalarda döngü veya çoklu bağlantı varsa, graf yapısı daha uygun olabilir. Dijkstra veya Floyd-Warshall gibi algoritmalarla en kısa yol hesaplanabilir.
5. **Paralel İşleme:**
   * Büyük veri kümelerinde sıralama veya arama işlemleri paralel işlem teknikleriyle hızlandırılabilir.

## Sonuç

Bu projede kullanılan veri yapıları ve algoritmalar, ihtiyaçlara uygun şekilde seçilmiş ve başarılı bir şekilde uygulanmıştır. Ancak daha verimli alternatifler de önerilebilir:

* HashMap ile daha hızlı veri erişimi sağlanabilir.
* BBST kullanılarak önceliklendirme yapılabilir.
* Graf yapıları, daha karmaşık rota yönetiminde avantajlıdır.

Bu önerilerle sistem daha hızlı ve etkili hale getirilebilir!

A computer screen shot of a computer

Description automatically generated