GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
VERİ YAPILARI DERSİ VİZE SINAVI

Adı – Soyadı – Numarası:

Soru 1: Aşağıdaki iki dizinin farkı nedir?

```
int[] sayilar1 = new int[5];
```

ifadesi statik bir dizi tahsis eder. heap üzerinde sabit boyutta (5 elemanlı) bir bellek bloğuna yer alır (allocate). tüm elemanlar int tipinin varsayılan değeri olan 0 ile ilklendirilir (initialize). Bu yaklaşım, derleme zamanında boyutun bilindiği veya çalışma zamanında sabit kalacağı senaryolarda tercih edilir çünkü bellek tahsisi tek seferliktir ve JVM'in garbage collector'ı tarafından öngörülebilir şekilde yönetilir. Performans açısından, bu dizi ardışık bellek yapısında olduğundan CPU ön belleğinde (cache) daha iyi kullanılabilir. ardışık erişimlerde (örneğin döngüyle tarama) cache miss oranı düşer ve işlemci pipeline'ı daha verimli çalışır.

```
int[] sayilar2 = {1, 2, 3, 4, 5};
```

ifadesi gerçek (literal) bir dizi ilklendiricisi (array initializer) kullanarak diziyi tanımlar. derleyici tarafından heap'te 5 elemanlı bir int[] nesnesi oluşturulur. Ancak bu kez elemanlar doğrudan belirtilen değerlerle doldurulur. Bu yazım (syntax), sadece tanımlama anında (declaration) geçerlidir. sonradan yeniden atama yapılamaz (örneğin sayilar2 = {6,7,8}; derleme hatası verir). Bunun yerine sayilar2 = new int[]{6,7,8}; şeklinde yeniden tahsis edilmesi gerekir. Performans açısından iki yaklaşım arasında çalışma zamanı farkı yoktur, çünkü her ikisi de aynı int[] sınıfının nesnelerini üretir; fark yalnızca ilklendirme maliyeti ve kod okunabilirliği düzeyindedir.

Sonuç olarak, new int[5] dinamik ilklendirme (sonradan doldurulabilir) ve bellek verimliliği (sıfırla başlatma hızlıdır) sunarken, {1,2,3,4,5} ise okunabilirlik ve hızlı veri atama avantajı sağlar. Büyük ölçekli sistemlerde, real-time veya low-latency uygulamalarda, new int[n] ile sıfır değeri ile ilklendirme + manuel doldurma tercih edilir.

Soru 2: Bağlı liste (LinkedList) yapısında elemanlara erişim dizilere göre neden daha yavaştır?

Bağlı liste (LinkedList) yapısında elemanlara erişim, dizilere göre daha yavaştır. çünkü bellek organizasyonu ve CPU mimarisi açısından farklı prensiplerle çalışır.

Diziler, ardışık (contiguous) bellek bloklarında saklanır; bu sayede arr[i] gibi rastgele bir erişim, başlangıç adresi + offset hesaplamasıyla O(1) zamanda ve tek bir makine talimatıyla gerçekleşir. Üstelik ardışık elemanlar aynı CPU cache line'ında (genellikle 64 byte) bulunduğundan, bir elemana erişildiğinde komşuları da otomatik olarak L1/L2 cache'e yüklenir (spatial locality avantajı).

Buna karşılık LinkedList, her düğümün (node) ayrı bir heap nesnesi olarak bellekte dağınık olarak yer alması nedeniyle, get(i) işlemi baştan i kez next pointer'ı takip etmeyi gerektirir. O(n) zaman karmaşıklığına sahiptir. her next referansı farklı bir bellek bölgesine işaret ettiğinden cache miss oranı dramatik şekilde artar. bir düğüm cache'e yüklendiğinde bir sonraki düğüm fiziksel olarak ilişkisiz olduğu için pipeline bozulur. branch prediction başarısız olur. her düğüm için int data (4 byte) yanında next referansı (8 byte) ve JVM nesne başlığı (12–16 byte) gibi bellek israfı (overhead) yaratır.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 7.11.2025

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
VERİ YAPILARI DERSİ VİZE SINAVI

Soru 3: Yığın (Stack) yapısı "geri alma (undo)" işlemleri için uygun mudur? Açıklayınız.

Yığın (Stack) yapısı, Son Giren İlk Çıkar (LIFO - Last In, First Out) prensibiyle çalışır. özellikle metin editörleri, grafik tasarım araçları veya veritabanı transaction sistemleri gibi uygulamalarda sıkça kullanılır. Temel mantık: Her kullanıcı eylemi (örneğin bir metin ekleme, silme veya nesne taşıma) yığına koyma (push) ile kaydedilir ve undo işlemi, en son eklenen eylemi çıkarma (pop) yaparak tersini uygular. örneğin eklenen metni silmek veya silineni geri getirmek. Bu yaklaşım, O(1) zaman karmaşıklığında sabit maliyetli bir işlem sağlar. stack pointer'ının tek bir increment/decrement ile en üst elemana erişmesi, hem CPU cache locality'sini korur hem de bellek parçalanmasını önler.

Soru 4: Yığın (Stack) ve Kuyruk (Queue) veri yapıları arasındaki farklar nelerdir?

Yığın (Stack) ve Kuyruk (Queue) veri yapıları, veri ekleme ve çıkarma sırasına göre farklılaşır. bu fark, hem algoritmik davranış hem de uygulama senaryoları açısından performans ve doğruluk etkileri doğurur. Stack, LIFO (Last In, First Out) prensibiyle çalışır; yani en son eklenen eleman (push) ilk çıkarılır (pop). Bu yapı, JVM stack frame yönetimi, fonksiyon çağrı yığını (call stack), geri alma (undo) mekanizmaları, derinlik öncelikli arama (DFS) ve parantez eşleştirme gibi geri dönüşlü, tersine çevrilebilir işlemler için idealdir.

Queue ise FIFO (First In, First Out) prensibiyle çalışır; ilk giren eleman (enqueue) ilk çıkar (dequeue). Bu, görev çizelgeleme (job scheduling), genişlik öncelikli arama (BFS), mesaj kuyrukları (RabbitMQ, Kafka), üretici-tüketici problemleri ve ağ paket tamponları gibi sıralı işleme gerektiren sistemlerde kullanılır.

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 7.11.2025

GİRESUN ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
VERİ YAPILARI DERSİ VİZE SINAVI

Soru 5: Aşağıdaki kod parçasında boş yerleri doldurunuz.

```
class Dugum {
    int veri;
    Dugum sonraki;
    Dugum(int veri) {
        this.veri = veri;
        this.sonraki = null;
    }
}
class BagliListe {
    Dugum bas;
    void basaEkle(int veri) {
        Dugum yeni = new Dugum(veri);
        yeni.sonraki = bas;
        bas = yeni;
    }
    void bastanSil() {
        if (bas != null)
            bas = bas.sonraki;
    }
}
```

```
class YiginYapisi {
    int[] dizi;
    int tepe;
    void push(int veri) {
        if (tepe == dizi.length - 1) {
            System.out.println("dolu!");
            return;
        }
        dizi[++tepe] = veri;
    }
    int pop() {
        if (tepe == -1) {
            System.out.println("bos!");
            return -1;
        }
        return dizi[tepe--];
    }
    int top() {
        if (tepe == -1)
            return -1;
        return dizi[tepe];
    }

}
```

Başarılar.. Dr. Öğr. Üyesi Sercan KÜLCÜ 7.11.2025