



**T.C.**  
**ONDOKUZMAYIS ÜNİVERSİTESİ**  
**MÜHENDİSLİK FAKÜLTESİ**  
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**  
**VERİ YAPILARI ÖDEV RAPORU**



**Ödev Başlığı: AVL ve BST (Binary Search Tree) Uygulaması**

**Ürünün İçermesi Gereken Özellikler: Performans Testi, Ekleme ve Arama Fonksiyonları**

**Ödev Kazanımları: Ağaç Veri Yapılarının Performans Üzerinde Farklılıkları**

**Yapılan Ürünün Özellikleri: Ağaçta Arama, Eleman Ekleme, Performans Test**

**Ürünün Kaynak Kodu ve Çalıştırılabilir Dosyaları (Linux ve Windows)  
“21060619\_alper\_karaca.zip” İsimli Sıkıştırılmış Klasörde Mevcuttur.**

# AVL Ağacı Sisteminin Gerçeklenmesi

**Struct Node \*newNode(int key){}**

```
struct Node *newNode(int key) {  
    struct Node *node = (struct Node *)  
        malloc(sizeof(struct Node));  
    node->key = key;  
    node->left = NULL;  
    node->right = NULL;  
    node->height = 1;  
    return (node);  
}
```

**Verileri işleyip sağ sol düğümleri oluşturan ardından yüksekliği de düzenleyen bir fonksiyon oluşturdum.**

**struct Node \*rightRotate(struct Node \*y){}**

```
struct Node *rightRotate(struct Node *y) {  
    struct Node *x = y->left;  
    struct Node *T2 = x->right;  
  
    x->right = y;  
    y->left = T2;  
  
    y->height = max(height(y->left), height(y->right)) + 1;  
    x->height = max(height(x->left), height(x->right)) + 1;  
  
    return x;  
}
```

**Ekleme işlemi ve denge için gereken sağa döndürme fonksiyonu.**

**struct Node \*leftRotate(struct Node**

```
struct Node *leftRotate(struct Node *x) {
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    return y;
}
```

**int getBalance(struct Node \*N){}**

```
int getBalance(struct Node *N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}
```

**Ağacı dengeler.**

```
struct Node *insertNode(struct Node *node, int key){}
```

```
struct Node *insertNode(struct Node *node, int key) {
    if (node == NULL)
        return (newNode(key));

    if (key < node->key)
        node->left = insertNode(node->left, key);
    else if (key > node->key)
        node->right = insertNode(node->right, key);
    else
        return node;

    node->height = 1 + max(height(node->left),
                           height(node->right));

    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);

    if (balance < -1 && key > node->right->key)
        return leftRotate(node);

    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}
```

Ağacı dengeleyip veri ekleme işlemini yapar.

```
struct Node *search(struct Node *root, int key) {}
```

```
struct Node *search(struct Node *root, int key) {  
    if (root == NULL || root->key == key)  
        return root;  
  
    if (root->key < key)  
        return search(root->right, key);  
  
    return search(root->left, key);  
}
```

Gönderilen ağaç içerisinde recursive işlemler kullanarak arama yapar. Bulursa o düğümü döndürür.

Main Fonksiyonu:

```
int main() {  
    struct Node *root = NULL;  
    clock_t start = clock();  
    for (int i = 0; i < 1000000; i++)  
    {  
        root = insertNode(root, i);  
    }  
  
    search(root, 1000000);  
    search(root, 100000);  
    search(root, 10000);  
    search(root, 1000);  
    search(root, 100);  
  
    printf("Gecen Sure: %.2fsaniye", (double)(clock() - start) /  
CLOCKS_PER_SEC);  
  
    return 0;  
}
```

Ağaca 1 milyon eleman ekledim ve aramalar yaptım.

# Uygulama Çıktıları

```
Windows PowerShell
[0 alpeerkaraca from Kirito][0s][0 RAM: 14/16GB][0 Friday at 8:53:54 PM][0 ?main = 0 ?2]
[~\Documents\Projeler\data-structures-assignment\Assignment-4]
.\avl_tree.exe
Ekleme Icin Gecen Sure: 0.35saniye
Arama Icin Gecen Sure: 0.00saniye
Toplam Gecen Sure: 0.35saniye
```

AVL Ağacına 1.000.000 eleman ekledim bu işlem 0.35 saniye sürdü.

- Arama kısmı anlamadığım biçimde aşırı kısa sürdü. Devamında test amaçlı 100.000.000 eleman ekledim ve bu süreç 53 saniye sürdü işlem süresinde %5 CPU ve 5GB'a yakın RAM tüketimi gözledim. Aramalarda bu sırada 1 saniye gibi bir süre aldı.
- Sistemimin 1 milyon eleman için iyi olduğu kanısındayım.
- Sistem özellikleri en son paylaşılacaktır.

# BST Yapısının Gerçeklenmesi

Ağacın içerisinde 43.000 Eleman Eklememin Sebebi Bundan Sonrasında Buffer Overflow hatası vermesi.

```
struct node *Create_Node(int item) {  
    struct node *temp = (struct node *)malloc(sizeof(struct node));  
    temp->key = item;  
    temp->left = temp->right = NULL;  
    return temp;  
}
```

Yeni düğüm içinde bellekte yer tahsisi yaptım.

`struct node *insert(struct node *node, int key){}`

```
struct node *insert(struct node *node, int key) {  
    // Ağaç boş ise yeni bir düğüm oluştur  
    if (node == NULL) return Create_Node(key);  
  
    // Traverse to the right place and insert the node  
    if (key < node->key)  
        node->left = insert(node->left, key);  
    else  
        node->right = insert(node->right, key)  
  
    return node;  
}
```

Ağaca recursive işlemler ile veri ekledim.



```
struct node *search(struct node *bst_tree, int s_key) {}
```

```
struct node *search(struct node *bst_tree, int s_key) {  
    if (bst_tree == NULL || bst_tree->key == s_key) return bst_tree;  
  
    // Değer ağaçtaki değerden büyükse sağa, küçükse sola git  
    if (bst_tree->key < s_key) return search(bst_tree->right, s_key);  
  
    // Değer ağaçtaki değerden küçükse sola git  
    return search(bst_tree->left, s_key);  
}
```

Recursive işlemler kullanarak verilen ağaçta istenen değeri ara ve mevcut ise düğümü döndür.

### Main Fonksiyonu

```
int main() {  
    clock_t begin = clock();  
    struct node *root = NULL;  
    for(int i=0; i<43000; i++)  
        root = insert(root, i);  
    //43.000 değerden sonra buffer overflow hatası veriyor.  
    printf("43.000 Eleman Ekleme İçin Geçen Sure: %.02f saniye\n", (double)(clock()  
- begin) / CLOCKS_PER_SEC);  
    clock_t start_search = clock();  
    search(root,25000);  
    search(root,12458);  
    search(root,12345);  
    search(root,42351);  
    search(root,25789);  
    search(root,39876);  
    search(root,14789);  
    search(root,15);  
    search(root,155);  
    search(root,41);  
    printf("Aramalar İçin Geçen Sure: %.lg saniye\n", (long double)(clock() -  
start_search) / CLOCKS_PER_SEC);  
    printf("Toplam Sure: %.02f saniye\n", (double)(clock() - begin) / CLOCKS_PER_SEC  
);  
}
```

Yukarıda da belirttiğim gibi en fazla 43 bin eleman ekleyebildim ve girilen değerleri arattım.

# BST İin ıktı:

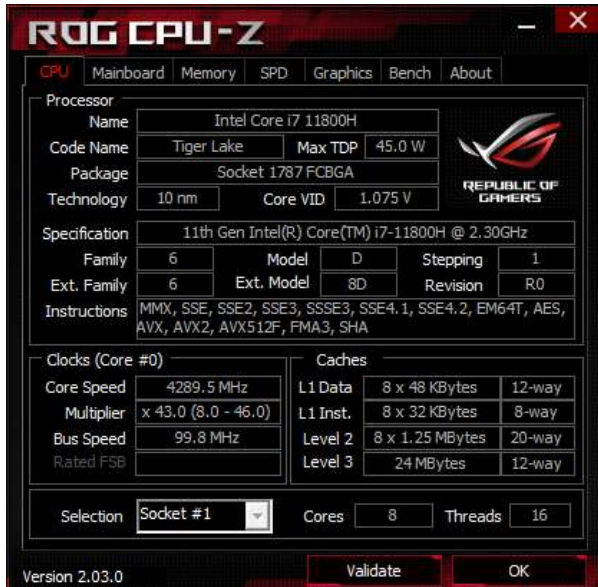
```
Windows PowerShell
[?] alpeerkaraca from Kirito [?] 0s [?] RAM: 14/16GB [?] Friday at 8:55:36 PM [?] main [?] ?2]
[?] [-\Documents\Projeler\data-structures-assignment\Assignment-4]
[?] .\BST.exe
43.000 Eleman Eklemek İcin Gecen Sure: 8.51 saniye
Aramalar İcin Gecen Sure: 3e-312 saniye
Toplam Sure: 8.52 saniye
```

Görüldüğü gibi daha az eleman eklememe rağmen işlem süresi oldukça fazla artmış durumda. Buradan da AVL ağacının gücü ortaya çıkmakta.

# Sistem Özellikleri

- Özetlemem Gerekirse:
- Intel I-7 11800-H 2.30GHz Base, 4.60GHz Turbo
- 1x16GB 3200MHz CL22 DDR4 Ram
- 1 TB PCI-E4.0 NVME SSD 3356MB/s Okuma / 1287 MB/s Yazma
- NVIDIA RTX 3050Ti, Intel UHD Grafik Kartı

## Detaylı Bilgiler:



ROG CPU-Z - Processor tab

**Processor**

Name	Intel Core i7 11800H		
Code Name	Tiger Lake	Max TDP	45.0 W
Package	Socket 1787 FCBGA		
Technology	10 nm	Core VID	1.075 V

**Specification**

11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz			
Family	6	Model	D
Ext. Family	6	Ext. Model	8D
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, AVX512F, FMA3, SHA		

**Clocks (Core #0)**

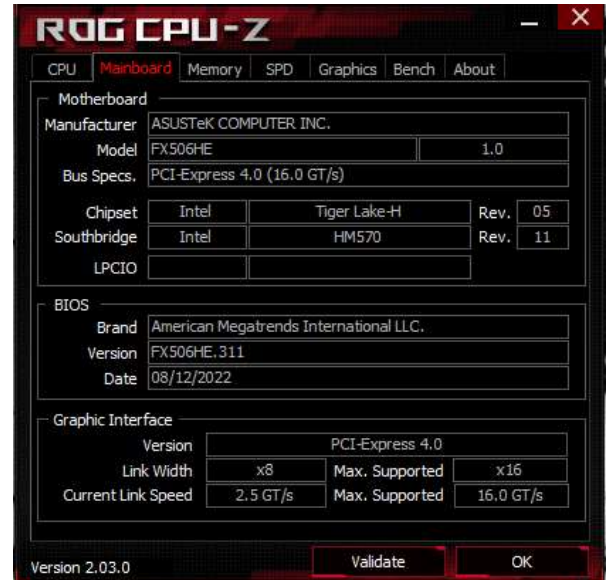
Core Speed	4289.5 MHz
Multiplier	x 43.0 (8.0 - 46.0)
Bus Speed	99.8 MHz
Rated FSB	

**Caches**

L1 Data	8 x 48 KBytes	12-way
L1 Inst.	8 x 32 KBytes	8-way
Level 2	8 x 1.25 MBytes	20-way
Level 3	24 MBytes	12-way

Selection: Socket #1 Cores: 8 Threads: 16

Version 2.03.0 Validate OK



ROG CPU-Z - Mainboard tab

**Motherboard**

Manufacturer	ASUSTeK COMPUTER INC.		
Model	FX506HE	Rev.	1.0
Bus Specs.	PCI-Express 4.0 (16.0 GT/s)		

**Chipset**

Chipset	Intel	Tiger Lake-H	Rev.	05
Southbridge	Intel	HM570	Rev.	11
LPCIO				

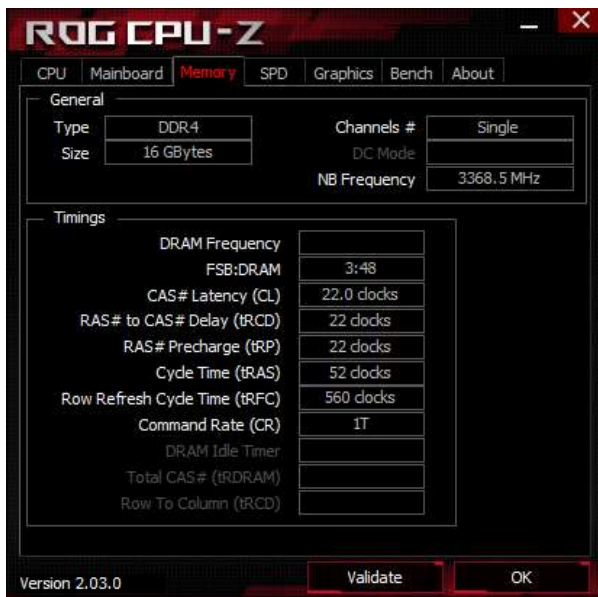
**BIOS**

Brand	American Megatrends International LLC.		
Version	FX506HE.311		
Date	08/12/2022		

**Graphic Interface**

Version	PCI-Express 4.0		
Link Width	x8	Max. Supported	x16
Current Link Speed	2.5 GT/s	Max. Supported	16.0 GT/s

Version 2.03.0 Validate OK



ROG CPU-Z - Memory tab

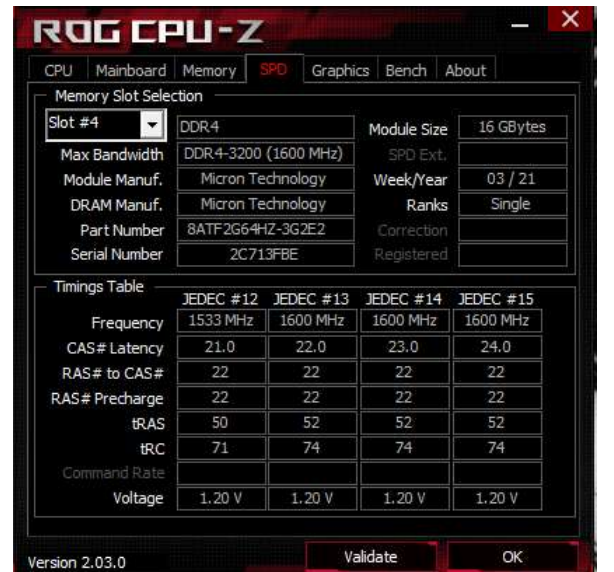
**General**

Type	DDR4	Channels #	Single
Size	16 GBytes	DC Mode	
		NB Frequency	3368.5 MHz

**Timings**

DRAM Frequency	
FSB:DRAM	3:48
CAS# Latency (CL)	22.0 clocks
RAS# to CAS# Delay (tRCD)	22 clocks
RAS# Precharge (tRP)	22 clocks
Cycle Time (tRAS)	52 clocks
Row Refresh Cycle Time (tRFC)	560 clocks
Command Rate (CR)	1T
DRAM Idle Timer	
Total CAS# (tRDRAM)	
Row To Column (tRCD)	

Version 2.03.0 Validate OK



ROG CPU-Z - SPD tab

**Memory Slot Selection**

Slot #4	DDR4	Module Size	16 GBytes
Max Bandwidth	DDR4-3200 (1600 MHz)	SPD Ext.	
Module Manuf.	Micron Technology	Week/Year	03 / 21
DRAM Manuf.	Micron Technology	Ranks	Single
Part Number	8ATF2G64HZ-3G2E2	Correction	
Serial Number	2C713FBE	Registered	

**Timings Table**

	JEDEC #12	JEDEC #13	JEDEC #14	JEDEC #15
Frequency	1533 MHz	1600 MHz	1600 MHz	1600 MHz
CAS# Latency	21.0	22.0	23.0	24.0
RAS# to CAS#	22	22	22	22
RAS# Precharge	22	22	22	22
tRAS	50	52	52	52
tRC	71	74	74	74
Command Rate				
Voltage	1.20 V	1.20 V	1.20 V	1.20 V

Version 2.03.0 Validate OK

# KAYNAKÇA

<https://bilgisayarkavramlari.com/2008/05/15/avl-agaci-avl-tree/>

<https://www.geeksforgeeks.org/binary-search-tree-data-structure/>