# 2D Arrays:

2D arrays are arrays with two sets of indexes. These arrays allow for more store information in a grid like pattern. These arrays are like the 1D arrays but with multiple 1D arrays within them. Below is an example of how a 2D array looks followed by code of how to populate one.

| Int[][] MyArr | MyArr[][0] | MyArr[][1] | MyArr[][2] |
| --- | --- | --- | --- |
| MyArr[0][] | 5 | 7 | 9 |
| MyArr[1][] | 1 | 2 | 3 |
| MyArr[2][] | 4 | 6 | 8 |

for(int i = 0; i <chaList.length;i++)

{

//loop for each column in 2d array (object creation)

for(int j = 0;j<chaList[i].length;j++)

{

//base setup for character object

Character player = new Character(name,stats,race

,Class,background,alignment,HP,SPD,AC);

//establish each variable of the character object

player.setStats();

player.setClass();

player.setRace();

player.setBackground();

player.setAlignment();

player.setName();

//assign object to the array

chaList[i][j] = player;

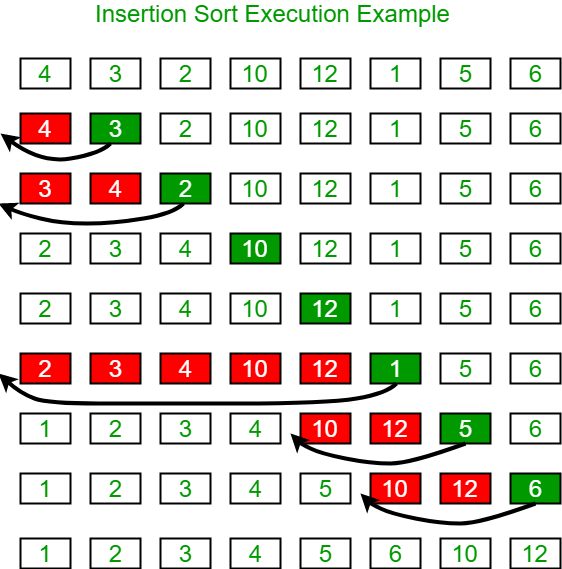
}

}

# Sorting Algorithms:

### Insertion Sort:

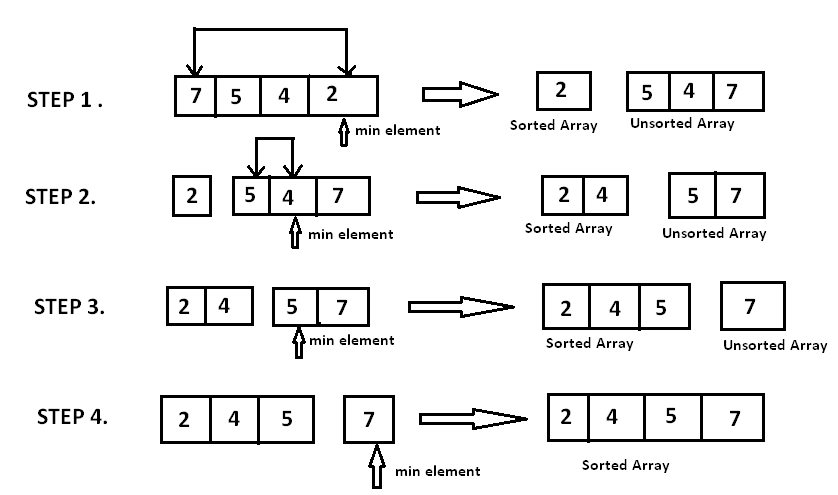
Insertion sort is a method of sorting an array that starts with the 0 index in slot 0 and compares the next index which is 1 against it. If index 1 is smaller than index 0 then the two swap places and you move on to the next index. The following index compares with the others to make sure that the value in index 0 is the lowest and reposition the others as needed. Below is a visual and a link to an animated explanation.



[https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiNq-zEtYeMAxWeDTQIHRFwGm0QwqsBegQIDBAG&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DOGzPmgsI-pQ&usg=AOvVaw0ExA52VpJJrxjnxfruRhy4&opi=89978449](https://www.youtube.com/watch?v=OGzPmgsI-pQ)

### Selection Sort:

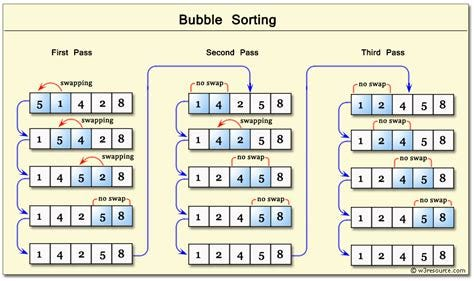
Selection sort is another form of sorting algorithm that takes the first index, index 0, and compares the rest of the indexes to it. If an index is smaller it assumes the new roll of index 0. This continues until the whole array has been checked and then you move to the next index until you have a fully sorted array. Below is an animation and visual aid.



[https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjKiM23tYeMAxViIjQIHVpkNJwQwqsBegQIDBAG&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DxWBP4lzkoyM&usg=AOvVaw3Rf6ZoN5svrqGiR99W\_QWD&opi=89978449](https://www.youtube.com/watch?v=xWBP4lzkoyM)

### Bubble Sort:

Bubble sort is a more time consuming sorting algorithm than the other two. Bubble sort compares two indexes at a time and swaps spots if the higher index is smaller than the lower index. Unlike the others which try to seek out the smallest first this method just moves on and then starts over until it is fully sorted through. Below is a visual aid and link to an animation for viewing.



[https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjimM34tIeMAxWOweYEHTeDI6oQwqsBegQICxAG&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DJP5KkzdUEYI&usg=AOvVaw3JeBUzUbcGr\_caNuI0amP7&opi=89978449](https://www.youtube.com/watch?v=JP5KkzdUEYI)

# Search Algorithms:

### Linear Search:

A Linear Search is a method used to find a specific element in an array. The specific element is known as a key and can be set by the user or by the programmer. How the linear search works is by taking the key and comparing it to each index of an array until it is found and outputting its position. Below is a snippet of code for a linear search.

public static int LinearSearch(int[] arr, int key)

{

for (int i = 0; i < arr.length; i++)

{

if (arr[i] == key)

{

return i;

}

}

return -1;

}

### Binary Search Tree:

A Binary Search Tree, or BST, is like the Linear search in the fact that it also requires a key for searching through an array. But unlike the Linear search, the BST requires a sorted array. This is due to the fact that this form of searching through the array starts from the middle and splits the array in half to see if the key is on one side or the other. This continues until either the array has fully been searched and the key doesn’t exist or until the key is found. The code for a Binary Search Tree is shown below along with a visual aid.

public static int BinarySearch(int[] arr, int key)

{

int low = 0;

int high = arr.length - 1;

while (high >= low)

{

int mid = (low+high) / 2;

if(key< arr[mid])

{

high = mid -1;

}

else if( key == arr[mid])

{

return mid;

}

else

{

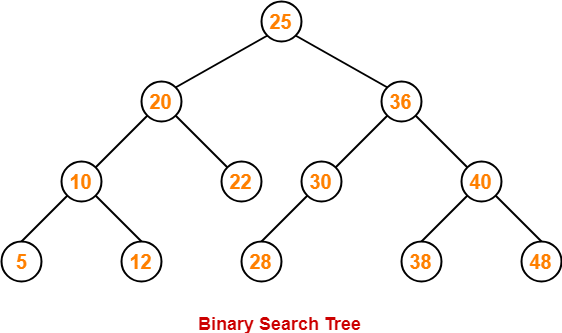
low = mid+1;

}

}

return -low -1;

}



# Software Development:

### Designing Classes:

When designing a class it is much easier to break the information down into smaller bite size pieces. When you are given a new problem to code you should think about what types of classes or methods you may need. What does each of those methods within those classes need to do along with what you need to output to the user of the program. The easiest way of doing this is with a UML diagram. UML diagrams are visual aids for programmers to break down what is needed in a class of code. Below is an example of a UML diagram plus the code it influenced.

| Character | Class name |
| --- | --- |
| -String chaName  -int[] stats  -String race  -String chaClass  -String Background  -String Align  -int health  -int armor  -int speed | Variables |
| +Character(String name, int[] attributes, String species, String Class, String background, String Alignment,int HP, int SPD, int AC)  + void setRace()  +void setName()  +void setBackground()  +void setAlignment()  +void setStats()  +getName()  +getChaClass()  +getBackground()  +getAlignment()  +getRace()  +getStr()  +getDex()  +getCon()  +getInt()  +getWis()  +getCha()  +getSPD()  +getArmor()  +getHP() | Methods |

class Character

{

//establish empty variables to be filled

String chaName;

int[] stats;

String race;

String chaClass;

String Background;

String align;

int health;

int armor;

int speed;

//Character constructor using variables from main class

public Character(String name, int[] attributes,String species,

String Class, String background, String Alignment,

int HP,int SPD, int AC)

{

//assigns the values from main to each associated variable in character class

this.chaName = name;

this.chaClass = Class;

this.Background = background;

this.align = Alignment;

this.race = species;

this.stats = attributes;

this.speed = SPD;

this.health = HP;

this.armor = AC;

}

//method for randomly establishing a race for the character

public void setRace()

{

//Min and max for random generation of base race selection

int mainMin = 1;

int mainMax = 9;

// selects a number between 1 and 9 for the race selection

int selectMain = new Random().nextInt(mainMax - mainMin

+1) +mainMin;

// dragonborn path

if (selectMain == 1)

{

//min and max for dragonborn type

int subMin = 1;

int subMax = 10;

//establish the random type of dragonborn the character will be

int selectSub = new Random().nextInt(subMax - subMin

+1) +subMin;

//assign stats and speed that all dragon born have

speed = 30;

stats[0] = stats[0] + 2;

stats[2]++;

//if statements to change race to the specific dragonborn type

if(selectSub == 1)

{

race = "Black Dragonborn";

}

if(selectSub == 2)

{

race = "Blue Dragonborn";

}

if(selectSub == 3)

{

race = "Brass Dragonborn";

}

if(selectSub == 4)

{

race = "Bronze Dragonborn";

}

if(selectSub == 5)

{

race = "Copper Dragonborn";

}

if(selectSub == 6)

{

race = "Gold Dragonborn";

}

if(selectSub == 7)

{

race = "Green Dragonborn";

}

if(selectSub == 8)

{

race = "Red Dragonborn";

}

if(selectSub == 9)

{

race = "Silver Dragonborn";

}

if(selectSub == 10)

{

race = "White Dragonborn";

}

}

//dwarf path

if (selectMain == 2)

{

//min and max for dwarf type

int subMin = 1;

int subMax = 2;

// selects between the two dwarf types

int selectSub = new Random().nextInt(subMax - subMin

+1) +subMin;

//establish the stat change and speed that all dwarfs have

stats[2]=stats[2]+2;

speed = 25;

//if statement for selecting between the two dwarf types

if(selectSub == 1)

{

race = "Hill Dwarf";

stats[4]++;

}

if(selectSub == 2)

{

race = "Mountain Dwarf";

stats[0]= stats[0]+2;

//gets more hp

health++;

}

}

//Elf path

if (selectMain == 3)

{

//min and max for elf sub type selection

int subMin = 1;

int subMax = 3;

//picks between the 3 elf types

int selectSub = new Random().nextInt(subMax - subMin

+1) +subMin;

//stat change and speed set that all elves have

speed = 30;

stats[1] = stats[1]+2;

//changing other stats and race based off of random number

if(selectSub == 1)

{

race = "Dark Elf";

stats[5]++;

}

if(selectSub == 2)

{

race = "High Elf";

stats[3]++;

}

if(selectSub == 3)

{

race = "Wood Elf";

//speed increase for the wood elf

speed = speed +5;

stats[4]++;

}

}

//gnome path

if (selectMain == 4)

{

//min and max for the gnome selection subrace

int subMin = 1;

int subMax = 2;

//picks between the two gnome types

int selectSub = new Random().nextInt(subMax - subMin

+1) +subMin;

//changes to speed and stats that all elves have

speed = 25;

stats[3] = stats[3]++;

//if statement to establish which gnome type the character is

if(selectSub == 1)

{

stats[1]++;

race = "Forest Gnome";

}

if(selectSub == 2)

{

stats[2]++;

race = "Rock Gnome";

}

}

//Half-Elf path

if (selectMain == 5)

{

//min and max for stat change selection for 2 extra skill points

int min = 0;

int max = 4;

//establishes speed and race name

race = "Half-Elf";

speed = 30;

int statChange;

int count =0;

//while loop for stat changes

while(count < 2)

{

//randomly selects stat that isnt Cha for stat boost

statChange = new Random().nextInt(max - min +1) +min;

stats[statChange]++;

count++;

}

}

//halfling path

if (selectMain == 6)

{

//min and max for halfling subrace

int subMin = 1;

int subMax = 2;

//selects the subrace number to assign to character

int selectSub = new Random().nextInt(subMax - subMin

+1) +subMin;

//boost to stats and establishment of speed for all halflings

stats[1] = stats[1]+2;

speed = 25;

//if statements to choose between the halflings

if(selectSub == 1)

{

race = "Lightfoot Halfling";

stats[5]++;

}

if(selectSub == 2)

{

race = "Stout Halfling";

stats[2]++;

}

}

//Half-Orc path

if (selectMain == 7)

{

//Establishes the race for half orc, speed, and stat changes

race = "Half-Orc";

speed = 30;

stats[0] = stats[0] +2;

stats[2]++;

}

//Human path

if (selectMain == 8)

{

//min and max for random number generator

int subMin = 1;

int subMax = 2;

//generates a 1 or 2 for human subtypes

int selectSub = new Random().nextInt(subMax - subMin

+1) +subMin;

speed = 30;

//if statements to establish race type and change stats

if(selectSub == 1)

{

race = "Human";

for (int count = 0; count < 6; count++)

{

stats[count]++;

}

}

if(selectSub == 2)

{

int min = 0;

int max = 5;

int statChange;

int count = 0;

System.***out***.println("");

while(count < 2)

{

statChange = new Random().nextInt(max - min +1) +min;

stats[statChange]++;

count++;

}

race = "Human Varient";

}

}

//tiefling path

if (selectMain == 9)

{

//establishes character as a tiefling and gives them the benefits that come with that

stats[5] = stats[5] +2;

stats[3]++;

speed = 30;

race = "Tiefling of Asmodeus";

}

}

//method for setting the random character name

public void setName()

{

//List of first names for male characters

String[] firstNameM = {"Olric","Amon","Fennic","Rowan","Petero","Magnus","Logan","Vincent","Xorni","Zilic","Noa",

"Urryn","Gunther","Yohan","Khan"};

//List of first names for female characters

String[] firstNameF = {"Rose","Charruni","Maven","Balka","Searen","Juleya","Luna","Wistina","Quinette","Freddrica",

"Ariel", "Vanya", "Guilda", "Erica", "Hydee"};

//List of last names for all characters(One left blank for those without a family name)

String[] lastName = {" Blackfoot"," Oakblood"," Von Everitt"," Nova",""," Mamon"," Tempest"," Gemic"," Swallow",

" Yggdra", " Ashborn", " Lionheart", " Genuva"," Locke", " Bloodmoon"};

//min and max index for selection of names from arrays

int min = 0;

int max = 14;

//int to be randomized for gender selection

int select;

//ints to be randomized for first and last name selection

int nameSelectFirst;

int nameSelectLast;

//place holder variables for last and first names

String lName;

String fName;

//min and max for gender selection

int minG = 1;

int maxG = 2;

//selects 1 for male characters or 2 for female characters

select = new Random().nextInt(maxG-minG+1)+minG;

//male name generation

if(select == 1)

{

nameSelectFirst = new Random().nextInt(max - min +1) +min;

nameSelectLast = new Random().nextInt(max - min +1) +min;

fName = firstNameM[nameSelectFirst];

lName = lastName[nameSelectLast];

//combines names into one

chaName = fName +lName;

}

//female name generation

if(select == 2)

{

nameSelectFirst = new Random().nextInt(max - min +1) +min;

nameSelectLast = new Random().nextInt(max - min +1) +min;

fName = firstNameF[nameSelectFirst];

lName = lastName[nameSelectLast];

//combines names into one

chaName = fName +lName;

}

}

//method for class selection

public void setClass()

{

//min and max for index of Jobs

int min = 0;

int max = 12;

//array of all Jobs(classes)

String[] Jobs = {"Artificer","Barbarian","Bard","Cleric","Druid",

"Fighter","Monk","Paladin","Ranger","Rogue","Sorcerer",

"Warlock","Wizard"};

//Randomly generates a number for class selection

int picker = new Random().nextInt(max - min +1) +min;

//if statements to establish character objects class, health, and armor

if(picker == 0)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 12+((stats[1]-10)/2);

}

if(picker == 1)

{

chaClass = Jobs[picker];

health = 12 +((stats[2]-10)/2);

armor = 10+((stats[1]-10)/2) +((stats[2]-10)/2);

}

if(picker == 2)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2);

}

if(picker == 3)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2)+2;

}

if(picker == 4)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2);

}

if(picker == 5)

{

chaClass = Jobs[picker];

health = 10 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2);

}

if(picker == 6)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 10+((stats[1]-10)/2)+((stats[4]-10)/2);

}

if(picker == 7)

{

chaClass = Jobs[picker];

health = 10 +((stats[2]-10)/2);

armor = 16;

}

if(picker == 8)

{

chaClass = Jobs[picker];

health = 10 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2);

}

if(picker == 9)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2);

}

if(picker == 10)

{

chaClass = Jobs[picker];

health = 6 +((stats[2]-10)/2);

armor = 10+((stats[1]-10)/2);

}

if(picker == 11)

{

chaClass = Jobs[picker];

health = 8 +((stats[2]-10)/2);

armor = 11+((stats[1]-10)/2);

}

if(picker == 12)

{

chaClass = Jobs[picker];

health = 6 +((stats[2]-10)/2);

armor = 10+((stats[1]-10)/2);

}

}

//method for setting a background to object

public void setBackground()

{

//min and max index for background selection

int min = 0;

int max = 15;

//selects a number to correspond with in pasts array

int select = new Random().nextInt(max - min +1) +min;

//array of all backgrounds

String[] pasts = {"Acolyte","Charlatan","Criminal","Entertainer",

"Folk Hero","Gladiator","Guild Artisan","Hermit",

"Knight","Noble","Outlander","Pirate","Sage","Sailor",

"Soldier","Urchin"};

//assigns the selected past to background

Background =pasts[select];

}

//method to assign an alignment

public void setAlignment()

{

// min and max alignment index numbers

int min = 0;

int max = 8;

//picks a number between 0-8 to make the index of the alignment

int select = new Random().nextInt(max - min +1) +min;

String[] Align = {"Lawful Good","Neutral Good","Chaotic Good"

,"Lawful Neutral","True Neutral","Chaotic Neutral",

"Lawful Evil","Neutral Evil","Chaotic Evil"};

//assigns selected alignment to align for object

align = Align[select];

}

//method for rolling your stats and assigning them

public void setStats()

{

//for loop to assign to all stats

for(int index = 0; index < stats.length;index++)

{

//min and max values of the roll

int min =1;

int max = 6;

//dice rolls to generate each stat

int dice1 = new Random().nextInt(max -min+1) +min;

int dice2 = new Random().nextInt(max -min+1) +min;

int dice3 = new Random().nextInt(max -min+1) +min;

int dice4 = new Random().nextInt(max -min+1) +min;

//array of the roll values

int[] rolls = {dice1,dice2,dice3,dice4};

//sorts the array to put the lowest in the first slot

Arrays.*sort*(rolls);

//adds all rolls besides the lowest roll to current stat location

stats[index] = rolls[1] +rolls[2] +rolls[3];

}

}

//returns the objects name

public String getName()

{

return chaName;

}

//returns the objects class

public String getChaClass()

{

return chaClass;

}

//returns the objects background

public String getBackground()

{

return Background;

}

//returns the objects Alignment

public String getAlignment()

{

return align;

}

//returns the objects Race

public String getRace()

{

return race;

}

//returns the objects Individual stats

public int getStr()

{

return stats[0];

}

public int getDex()

{

return stats[1];

}

public int getCon()

{

return stats[2];

}

public int getInt()

{

return stats[3];

}

public int getWis()

{

return stats[4];

}

public int getCha()

{

return stats[5];

}

//returns the objects speed

public int getSPD()

{

return speed;

}

//returns the objects armor rating

public int getArmor()

{

return armor;

}

//returns the objects HP

public int getHP()

{

return health;

}

}

### Test Driven Development(TDD):

Test Driven Development, or TDD, is the process of sending different hypotheticals at your code to make sure you have responses that don’t just break the program. These could be invalid inputs, invalid case sensitivity, or no input found. These are broken up into preconditions, postconditions and notes for what can happen. Below is a few examples of these test cases of TDD:

| Preconditions | Postconditions | Notes |
| --- | --- | --- |
| The user entered “Sam” when asked to give their age. | Incorrect variable type.  Try to enter a number. | Message is displayed if the user enters the wrong variable type. |
| Reading from file a species type of “parrot” into if statement of input == “Parrot” | Something wasn’t read correctly.  Please check case sensitivity | Must add || input == “parrot” into if statement to allow for both options. |
| Arr.length = 5  System.out.println(arr[5]) | Index is outside of array range.  Please look for a number between 0 and 4 | Code is looking for a value in index 5 which doesn’t exist because the array is 5 in length and arrays start at index 0. |

# Big O:

Big O notation is an algorithmic theory about how to optimize code to know which method or Algorithm to use to save on time and resources. The Growth rates of Big O notation are as follows:

| Constant | O(1) |
| --- | --- |
| Logarithmic | O(log n) |
| Linear | O(n) |
| Log Linear | O(n log n) |
| Quadratic | O(n2) |
| Polynomial | O(nc) |
| Exponential | O(2n) |

Some examples listed throughout this document of Big O that we use is Selection Sort, Bubble Sort, and Insertion Sort which are all examples of Quadratic growth rates. Listed below is a visual for the growth rates of Big O Notation.

