Asian MathSci League, Inc (AMSLI)
Partner: National Olympiad in Informatics Philippines (NOI.PH)
Email address: amsliphil@yahoo.com / ask@noi.ph
Contact Nos: +632-9254526 +63906-1186067

Student Copy
AIEP Scratch 2 Session 7A

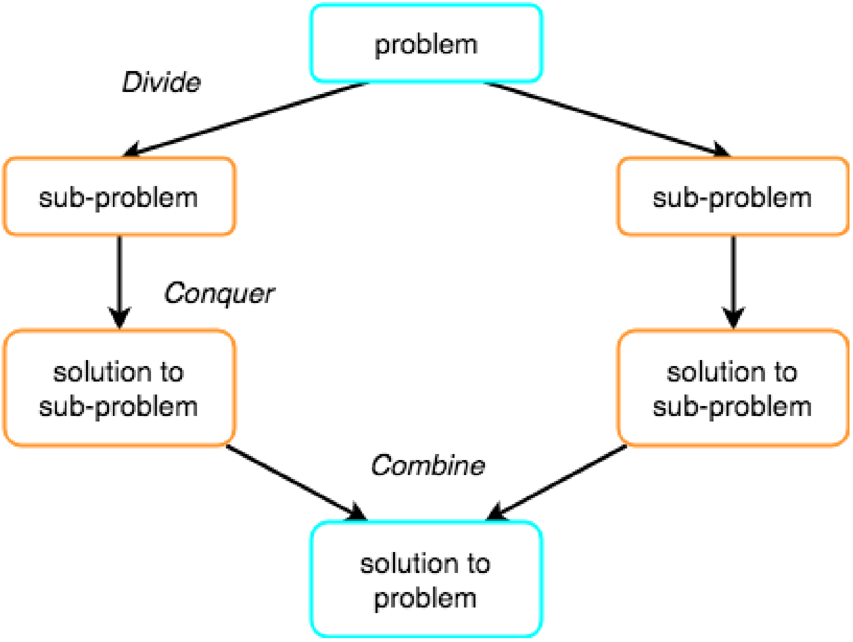
DIVIDE AND CONQUER

"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."

~ Bill Gates

In the previous AIEP sessions, we focused on formulating algorithms and writing programs that can perform a given task correctly. However, real life is often more complicated than just doing things correctly. We have to do things fast; thus, we have the adage "speed and accuracy." In computer science, this is known as the notion of EFFICIENCY.

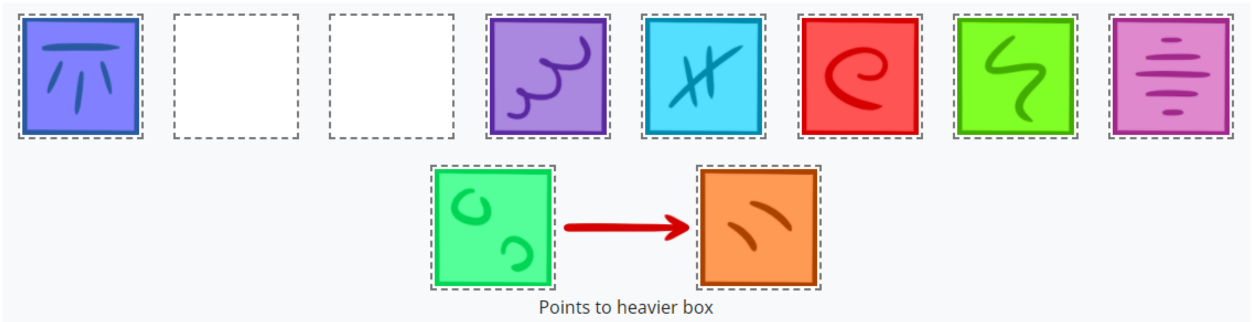
But how do we even make efficient algorithms in the first place? This is a tough question, and entire domains of computer science and mathematical research are devoted to this. In this handout, we are going to learn a common technique used to speed up algorithms: DIVIDE AND CONQUER.



https://www.studytonight.com/data-structures/merge-sort

ACTIVITY: Sorting... Without Numbers?

Suppose that we have some boxes of different weights, and we have to arrange them from lightest to heaviest. **However, there is a twist: their actual weights are hidden!** Luckily, we have a "weighing seesaw" — which we can use to determine the heavier between two boxes.



This is the crux of the sorting problem on the following website: https://csfieldguide.org.nz/en/interactives/sorting-algorithms/.

Challenge: Can you minimize the number of comparisons needed? Ideally, the number of comparisons should be around 16.

Discussion: Mergesort

This activity gives us an idea as to how computers "struggle" with sorting values. Unlike us, machines can only rely on results of comparisons (the "weighing seesaw")[1], coupled with a well-defined sequence of steps. However, algorithms vary greatly in terms of speed[2]:

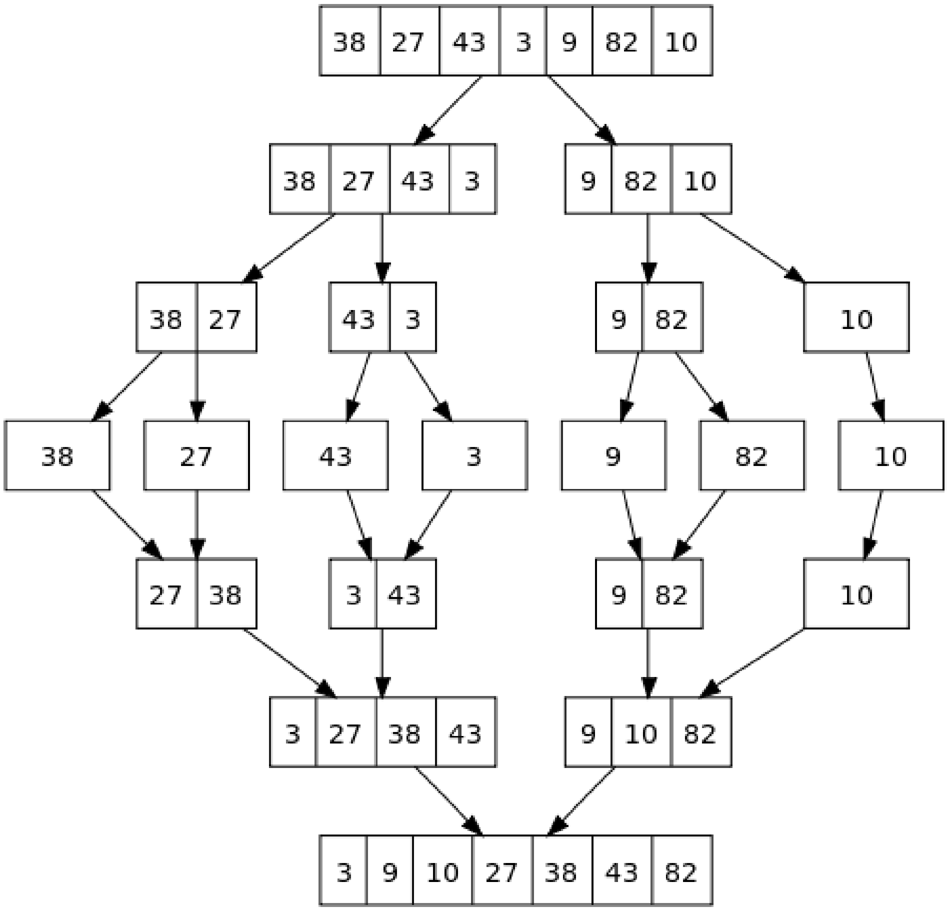
a. 😔 **Bubble sort** takes **half an hour** to sort 1048576 numbers.
b. 😄 **Shellsort** takes around **half a second** to sort the same set of 1048576 numbers.
c. 😎 **Quicksort** takes only a **quarter of a second** to sort this set of 1048576 numbers.

[1] Technically, there are sorting algorithms that are not based on comparisons. These include bucket sort, radix sort, and counting sort. However, there are special conditions that must be met before they can be applied.
[2] These are based on actual data that our group gathered for a data structures and algorithms course. The code was written in C, one of the fastest high-level programming languages.

One of the most efficient sorting algorithms is MERGESORT. Developed by the Hungarian-American computer science pioneer John von Neumann in 1945, it is a good example of a divide-and-conquer algorithm:

a. Divide: We repetitively (recursively) divide the list of values into halves until we get single-element sublists.
b. Conquer: We merge the sublists, taking two at a time, with the goal of producing sorted sublists. We continue doing this until only a single list is left — voila, sorted!

Mergesort is a rather advanced example of a recursive procedure (recall from Session 4B: a recursive procedure is one that calls itself). A visualization of the algorithm is shown below *(the "divide" part is the upper half whereas the "conquer" part is the lower half)*:



https://dotnettutorials.net/lesson/merge-sort-algorithm-in-csharp/

ACTIVITY: Country Search Engine

Load the skeleton Scratch file Country Search Engine_For Students.sb3. Create four lists and import their contents from the corresponding CSV files[3]:

a. `countries` - countries.csv
b. `capitals` - capitals.csv
c. `currencies` - currencies.csv
d. `languages` - languages.csv

CSV, or comma-separated values, files are special types of text files that are used to store data in a simple format that can be recognized by and imported into almost all programs. The contents of a CSV file can be stored into a Scratch list by right-clicking the list on the stage and choosing Import.

Discussion: Binary Search

In computer science, there are two main ways to search the contents of a list:

Type of Search	Description	Requirement
Linear Search	Go through each item inside the list sequentially, starting with the first item and stopping when the key is found (or the entire list has been searched).	Works for both unsorted and sorted lists
Binary Search	This is a divide-and-conquer algorithm that is highly efficient for sorted lists.	Works only for SORTED lists

The `item # of __ in __` block in Scratch uses linear search, which works for all types of lists. However, if the list is already sorted, is it necessary to go through each item one by one? For example, if we are looking for a word in the dictionary or in a book's index, we do not scan all the entries, one by one, from the start. That would be way too slow.

[3] Data taken from https://www.bankexamstoday.com/2019/06/countries-capital-currency-and-languages.html
The usage is intended only for illustrating binary search and related programming concepts. The completeness and veracity of the claims and information therein are outside the scope of the exercise.

BINARY SEARCH comes to the rescue! The idea of this algorithm is as follows:

a. Get the middle of the list as our initial "guess."
b. If it is equal to the item that we are looking for, then we are finished.
c. If it is less than the item that we are looking for, then this target item belongs to the upper half. *The lower half is just thrown away.*
d. If it is greater than the item that we are looking for, then this target item belongs to the lower half. *The upper half is just thrown away.*
e. Continue doing this to the remaining half until the item is found.

Let us have a walkthrough with a sample list of values (names of programming languages). In implementing binary search, we need to have three variables: `left`, `right`, and `middle`.

1	2	3	4	5	6	7	8	9	10
Ada	Clojure	Go	Haskell	Java	Perl	Prolog	Python	Ruby	Swift
`left`				`middle`					`right`

Suppose we want to get the index of Perl. Index refers to the position of an item in the list. The element at the middle is Java, but Java is less than Perl, so we proceed to searching only the upper half *(the lower half is grayed out).*

1	2	3	4	5	6	7	8	9	10
Ada	Clojure	Go	Haskell	Java	Perl	Prolog	Python	Ruby	Swift
					`left`		`middle`		`right`

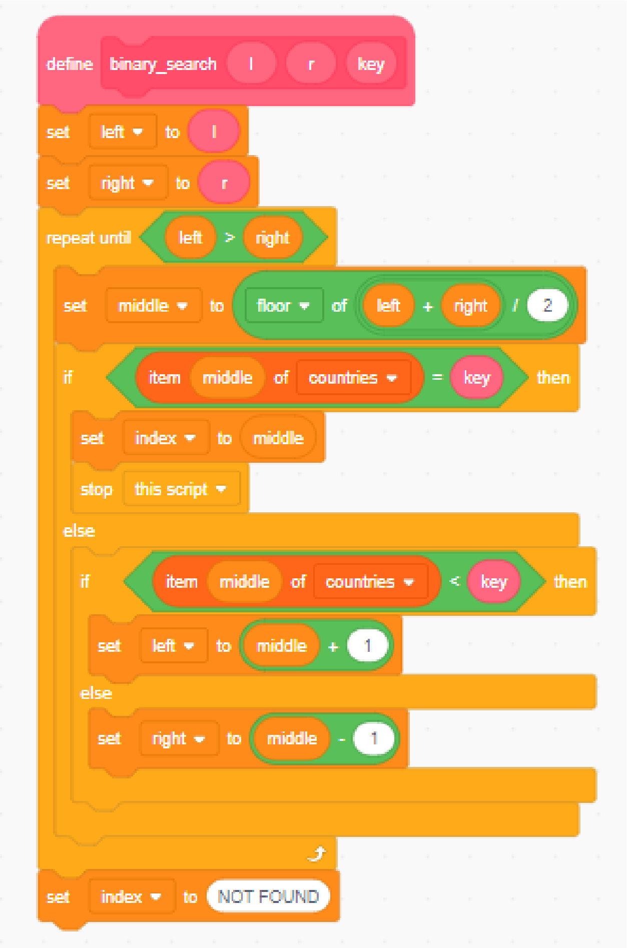
The element at the middle is now Python, but Python is greater than Perl, so we proceed to searching only the lower half:

1	2	3	4	5	6	7	8	9	10
Ada	Clojure	Go	Haskell	Java	Perl	Prolog	Python	Ruby	Swift
					`left` `middle`	`right`			

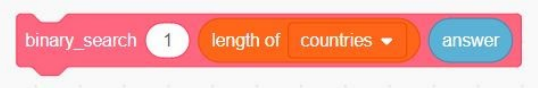
The element at the middle is now Perl, and Perl is equal to Perl — voila, the index is 6!

Can you figure out how the values of **`middle`, `left`,** and **`right`** are updated every time?

The implementation of the algorithm in Scratch is as follows (`countries` is our list):



```
define binary_search (l) (r) (key)
set [left] to (l)
set [right] to (r)
repeat until <(left) > (right)>
    set [middle] to ([floor] of (((left) + (right)) / (2)))
    if <(item (middle) of [countries]) = (key)> then
        set [index] to (middle)
        stop [this script]
    else
        if <(item (middle) of [countries]) < (key)> then
            set [left] to ((middle) + (1))
        else
            set [right] to ((middle) - (1))
set [index] to [NOT FOUND]
```



```
binary_search (1) (length of [countries]) (answer)
```

Important Notes:

The parameters `l` and `r` are initially set to 1 and to the length of the list, respectively, since our goal is to search the entire list.

`key` is the formal term for the item that we are looking for. In the sample My Block call above, it is set to `answer` (but this can, of course, vary).

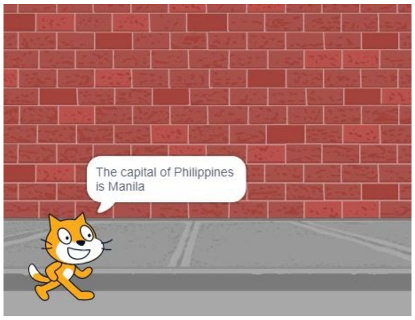
For pedagogical purposes, `index` is set to `NOT FOUND` if a key is not in the list. In actual programming practice, we usually raise something called an exception.

Going Back to the Activity

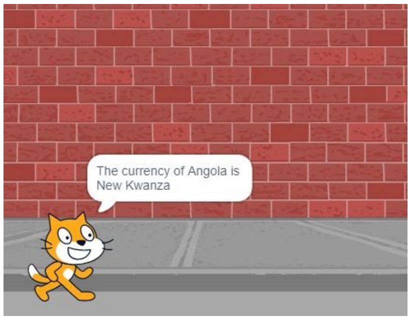
Try to complete the code in the skeleton file Country Search Engine_For Students.sb3 and accomplish the following tasks using binary search:

1. Get the **capital** of a country.
2. Get the **currency** of a country.
3. Get the **languages** of a country, as well as the **number of languages**. The format for the output should be either of the following:
    a. ______ has only 1 language: _______.
    b. ______ has _____ languages: _______.

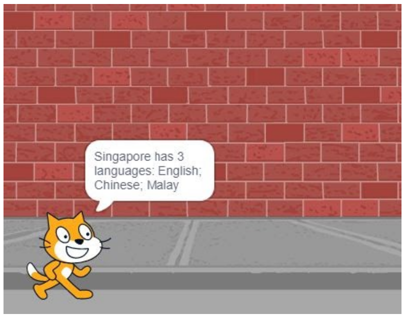
Needless to say, you are not allowed to use the built-in `item # of __ in __` block in Scratch for two reasons: (a) it is too slow and (b) it defeats the purpose of this discussion. If a country is not in our list, the sprite should say `“No search results found”`. Sample outputs of the program are shown below:



The capital of Philippines is Manila



The currency of Angola is New Kwanza

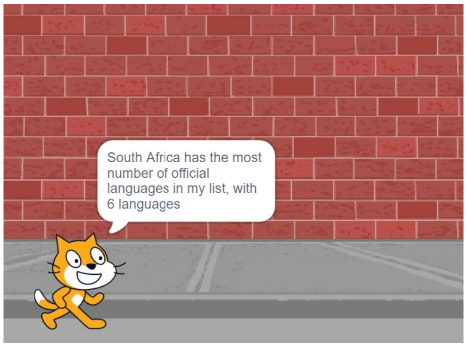


Singapore has 3 languages: English; Chinese; Malay

The fourth task __**cannot**__ **be accomplished using binary search** ***(Why?)***. It is instead meant to be an application of the previous lesson on lists (Session 5):

4. Get the country with the **most number of official languages** (as well as the number of languages).

   The expected output is seen in the adjoining figure on the right.



South Africa has the most number of official languages in my list, with 6 languages

QUESTIONS for #Unplugged Discussion

1. How is the binary searching procedure related to binary search trees[4]?
2. When would it be better (a) to just perform a **simple linear search** or (b) to **sort** the values then perform a **binary search**? What are the tradeoffs of each approach?

-- return 0; --

[4] Learn more about binary search trees: https://visualgo.net/bn/bst?slide=1