Sequential and Binary Search

In this activity, you will learn about sequential searches and binary searches. Each algorithm will be used to determine if a list contains a certain item. One of these searches requires the data to be sorted, the other does not. By the end of the activity, you will have coded both search types. You will be able to determine the maximum number of comparisons each must make before determining a list does not contain an element.

Objectives:

Programming Objectives:

- Use selection statements to make decisions.
- Use iteration to repeat a process.
- Use lists to store data.
- Import lists from a spreadsheet into Python code.
- Use library functions to generate random integers.

Key AP Computer Science Principles Standards:

- Represent a value with a variable (AAP-1.A)
- Represent a list or string using a variable. (AAP.1 C)
- For selection, write conditional statements (AAP-2 H)
- For iteration, write iteration statements (AAP-2.K)
- For list operators, write expressions that use list indexing and list procedures (AAP-2 N)
- For algorithms involving elements of a list, write iteration statements to traverse lists (AAP-2.O)
- Select appropriate libraries or existing code segments to use in creating new programs (AAP-3.D)
- Linear search or sequential search algorithms check each element of a list, in order, until the desired value is found or all elements in the list have been checked. (AAP-2.O.5)
- The binary search algorithm starts at the middle of a sorted data set of numbers and eliminates half of the data; this process repeats until the desired value is found or all elements have been eliminated (AAP-2.P.1)
- Data must be in sorted order to use the binary search algorithm. (AAP-2.P.2)
- Binary search is often more efficient than sequential/linear search when applied to sorted data.
(AAP-2.P.3)

This document contains two activities.

Activity 1: Sequential Search

Students use **function** in the TI-System library to import lists from a spreadsheet page.
Students use selection and iteration to perform a Sequential Search through a list.

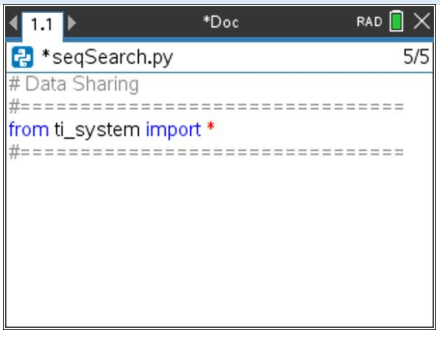
Activity 2: Binary Search

Students use **function** in the TI-System library to import lists from a spreadsheet page.
Students use selection and iteration to perform a Binary Search through a sorted list.
If students have access to a TI-Innovator Hub, they will add lights and sounds to the binary search.

Activity 1: Sequential Search

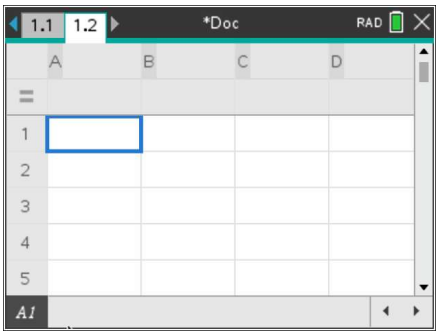
1. Create a new program named “seqSearch”.

   Choose Data Sharing for the default type.



*You can name your project “seqsearch” all in lower case. However, using capital letters for the S makes it easier to read the name of the file. Often, programmers capitalize the first letter of each new word in a variable name. This form of naming is referred to as “camel case”. It doesn’t change how the program runs; it does however make it easier for the programmer to read.

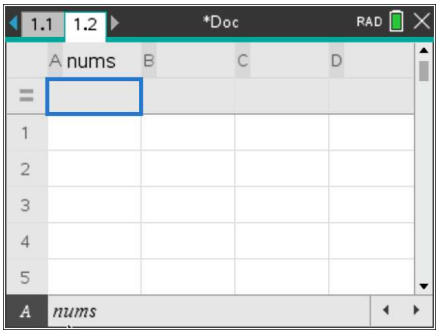
2. In this project, you will investigate a search method named “Sequential Search”. Using this method, your program will look for a specific number, one item at a time until it is found in the list, or the end of the list is reached without finding it.



A simple way to create a list of numbers is to add them to a spreadsheet page.

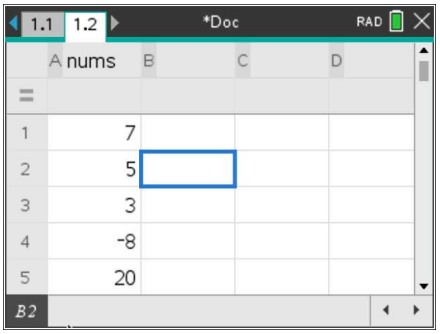
Add a Lists and Spreadsheet page.

3. Create a list named nums.

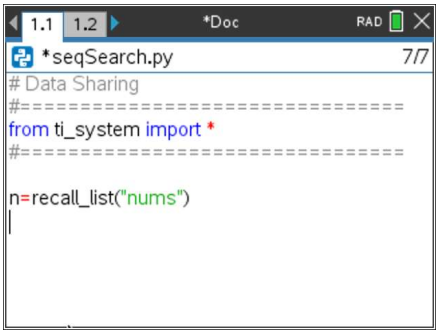


To create the list, type “nums” in column list A.

4. Add the following 5 numbers to your list:
   7, 5, 3, -8, 20



5. Go back to your Python page.



To import the list, you will use the feature recall_list
   Menu > More Modules > Ti-System > recall_list

You can store the list in any variable name you like. For ease of typing, we will call the list in Python n. The line:

n = recall_list(“nums”)

imports the list nums from the spreadsheet and stores it in a variable named n.

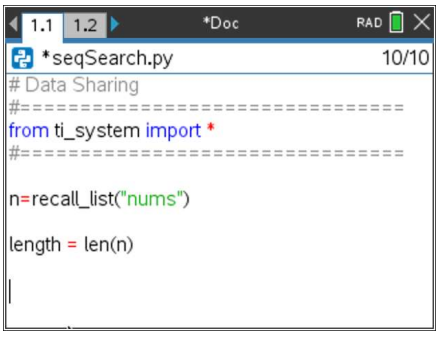
Your program seqSearch.py from here on out will only recognize the variable name n, the list “nums” from the spreadsheet won’t have any meaning in the code.

6. Let's verify your list was imported correctly.

   Find the length of the list, store it in a variable named length:

   Menu > Built-Ins > Lists > len

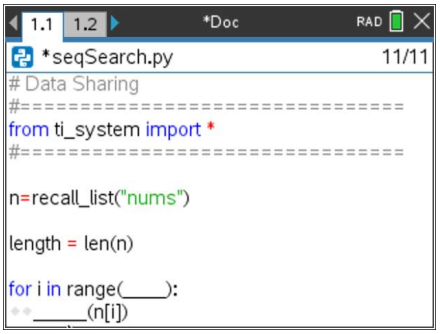
   length = len(n)



7. Use a loop to print each item in the list.
   Fill in the blanks, then code your project:

   for i in range(_____):

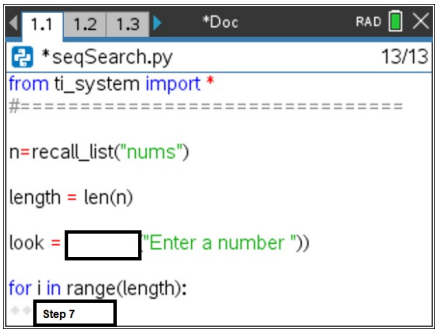
   _____(n[i])



8. Run your code.

   Did it print all five numbers in the same order you typed them on the spreadsheet?

9. You need to ask the user for an integer to look for in the list.
   Ask the user for an integer, store this value in a variable named look.



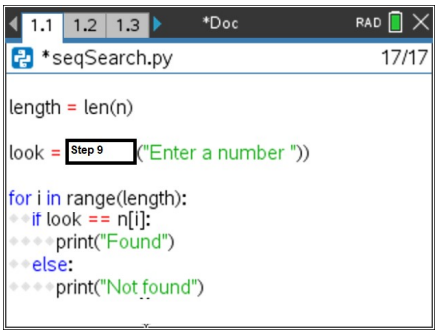
10. Now to add an if statement.

    If the number is found, print "Found".
    If the number is never found, print "Not Found"

    Remove the print statement.

    Add the code:

```
if look == n[i]:
    print("Found")
else:
    print("Not found")
```



11. Execute your program twice.

    The first time, look for -8. It is in the list, so you should see "Found"

    Does it only print Found?

    The second time, look for 100. It isn't in the list, so it should print "Not found"

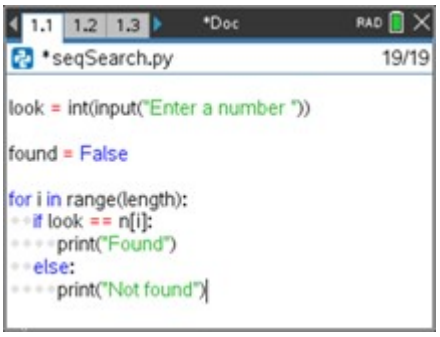
    Does it only print Not Found?

12. To make the code only print once, you need to add a Boolean variable.

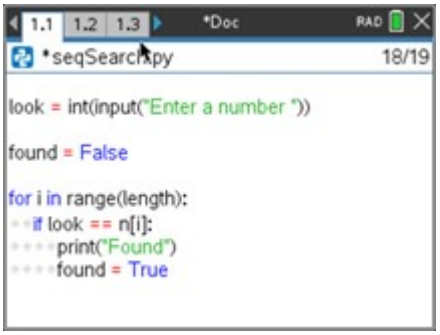
    Boolean variables can hold only two values; True or False.

    To begin with, we will say found = False

    Add this before your loop.



13. Inside the loop, we want it to print "Found", then change the Boolean value to True. Modify your code so it no longer prints "Not found", but instead changes the Boolean variable found to True when found.



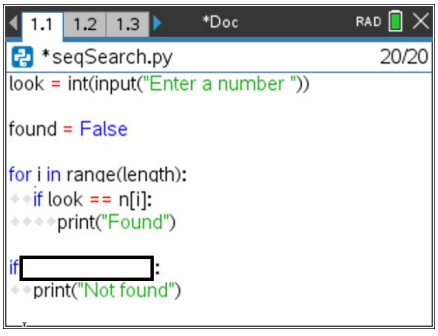
14. Execute code. Again, do a search for -8 which is in the list.
    Then search for 100 which is not in the list.

    Does it only print "Found" once for -8?

    What does it print for 100?

15. After the loop, the program should print "Not found" only if the variable found is still False.

    Add an if statement after the loop to print "Not found" if the variable found is False.



16. Execute your program.

    Verify it says “Not found” when you enter values that are not in the list.

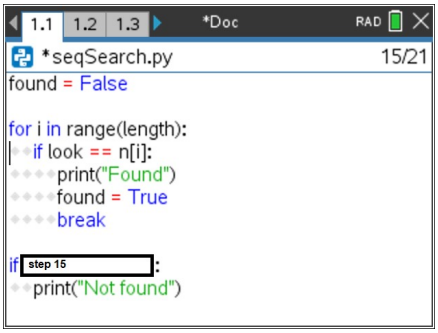
17. How efficient is your code if there are 500 items in your list?

    Currently, how many comparisons will be made if the item is in a list of 500?

    How many comparisons will be made if the item is not in the list 500?

18. Does it make sense to continue to look if the item was found? No! That is wasted time.

    Add a break to your if statement.
    If the item is found, break out of the loop



19. How can you verify this works correctly?

    Add a print statement to your loop such that the print statement prints every number that was checked.

20. Verify your code.

    If you enter the number 5, it should only print 7 and 5 before it prints “Found” and exits the loop.

    If you enter 100, it should print all 5 numbers before it prints “Not Found”.

21. With this modification,

    How many comparisons will be made if the item is in a list of 500?

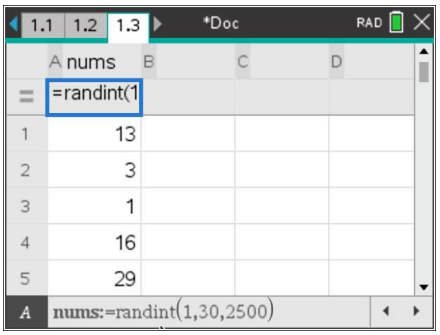
    How many comparisons will be made if the item is not in the list 500?

22. You've made your code more efficient. But how efficient is it really?

    Let's try it out with a list of 2500 items.

    Go back to your spreadsheet page. In the formula column (the one with the =) type the command randint(1,3000,2500).

    This will create 2500 random numbers from 1 to 3000.



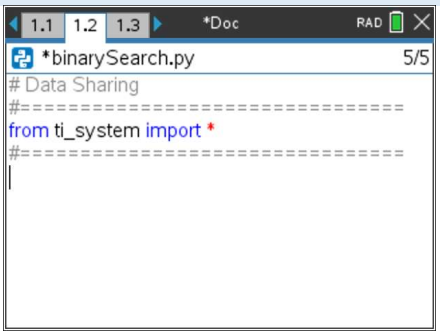
23. If possible, run this program on computer software and on the handheld calculator. What do you notice?

Activity 2: Binary Search

1. Add a new Python page to your document.

   Name the program "binarySearch".

   Choose Data Sharing for the default type.



2. A Binary Search is different from a Sequential search. Before we get started, let's remember what a sequential search does.

   a. Explain how a sequential search works:

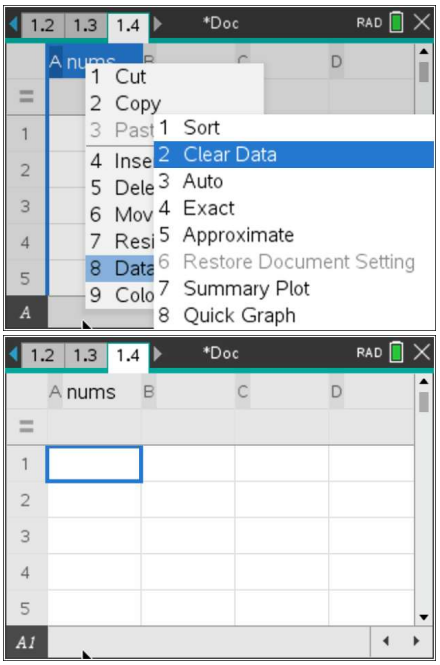
   b. If there are 100 items, how many comparisons does it make if the item is in the list?

   c. If there are 100 items, how many comparisons does it make if the item is not in the list?

3. To begin, a Binary Search requires your items to be in order.

   Go back to your spreadsheet page (if you have one) and delete the old data. If you don't have a spreadsheet page on this project, add one.

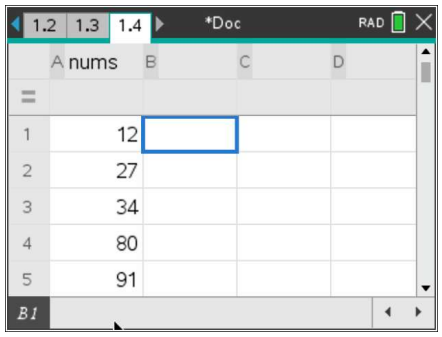
   Tip: To clear out old list values, right click name of the list.
   Data > Clear Data



4. Add the numbers 12, 27, 34, 80, 91, 120, 141, 171, and 172.

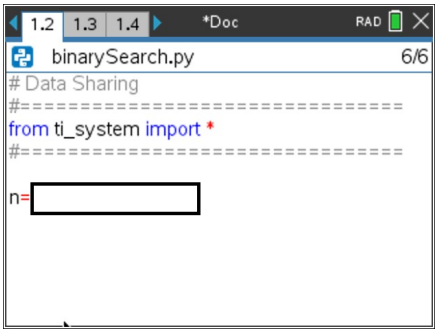
   Make sure you enter them in order from least to greatest.

   *Make sure the list name is nums.



5. Do you recall how to import the list?
   If need be, look back at your code from the sequential search.

   Store your nums list in a variable named n.



6. A Binary Search looks at the number in the middle.

12
27
34
80
91  If this is the number, you are done
120
141
171
172

If it isn't the middle number, determine if it is in the lower half or the upper half.

If it is in the lower half,
It is in item 0 through middle – 1

12
27
34
80

If it is in the upper half,
it is in item middle + 1 through the end

120
141
171
172

Now repeat the process, find the middle of the new section. If it is the middle item, you're done, otherwise, "discard" half the list.

7. The process could look something like this. Is 34 in the list?

12
27
34
80
91  No
120
141
171
172

12
27  no
34
80

34  found in 3
80

Or….is 150 in the list?

12
27
34
80
91  No
120
141
171
172

120
141  no
171
172

171  no
172

172  no, searched all items in 4 steps

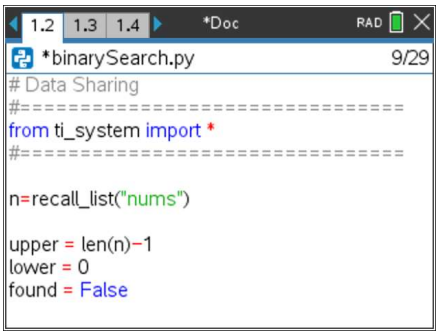
Notice, unlike the sequential search, the binary search does not have to compare the value to each item in the list. Because the list is in order, it can "discard" half the list in the search after looking at each new middle.

8. Let's create the code.

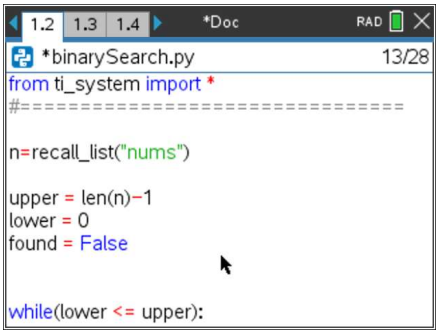
   To start, you need to know the upper and lower boundary for the part of the list that is still "in play". Initially, these values will be 0 as the minimum and one less than the number of items in the list as the maximum.

   Add the code:

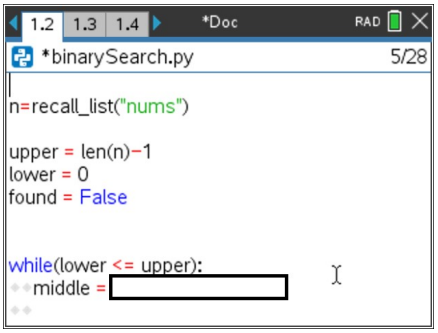
   ```
   upper = len(n)-1
   lower = 0
   found = False
   ```



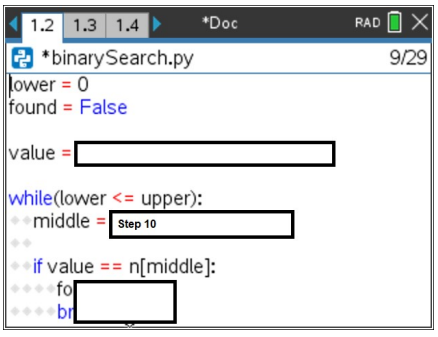
9. While the lower bound is less than or equal to the upper bound, search for the item.



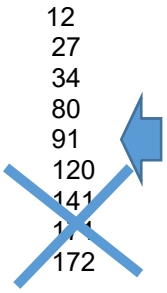
10. The middle number will be the integer value of the upper plus the lower cut in half. Make sure you turn this value into an integer, your index value must be an integer.



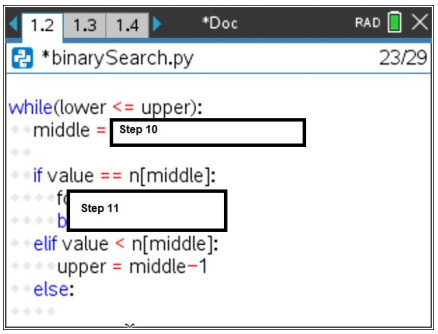
11. The user needs to enter a value to search for in the list. Ask the user before the loop for an integer value to search for. In the loop, check to see if the value matches the middle item in the list. If so, print found, change the variable found to True and break out of the loop.



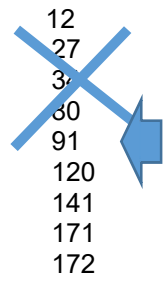
12. If the value comes before the middle item,



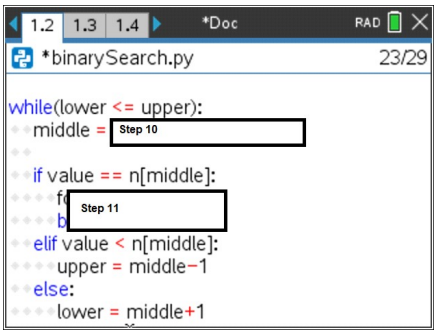
Change the upper bound to be the middle - 1



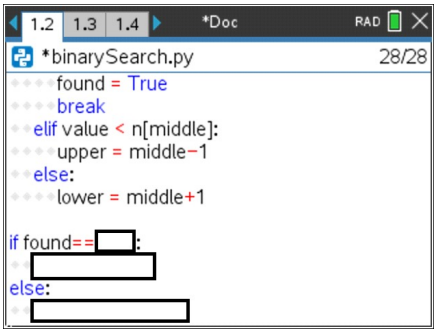
13. If the value came after the middle item,



Change the lower bound to be the middle + 1



14. Last but not least, print “Found” if the item was found, otherwise print “Not Found”



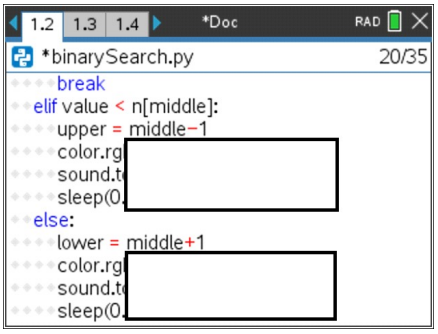
15. Execute your program several times. Does it work for all values in your list including the endpoints?

16. If you have a TI-Innovator Hub, do this step. If not, skip to step #17.

Let’s spice this project up a bit!!

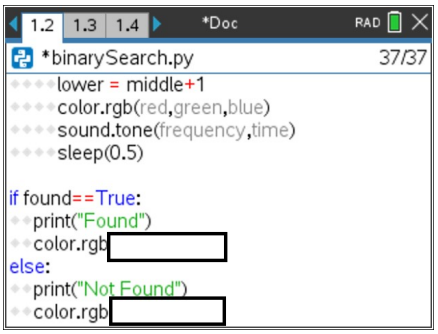
You can see the print word “Found”, but how exciting is that?

Let’s add in some lights and sounds.
If the value is lower than current middle item,
play a sound and set a color for ½ a second.
If the value is higher than the current middle item,
play a different sound and color for ½ a second.



At the top of your code, import the hub library.

Inside the loop, add the lines of code to set the LED and play a tone.



At the end of your code, set the LED to one color if the item was found, another color if it wasn't found.

17. Let's say you have the following items in a list.

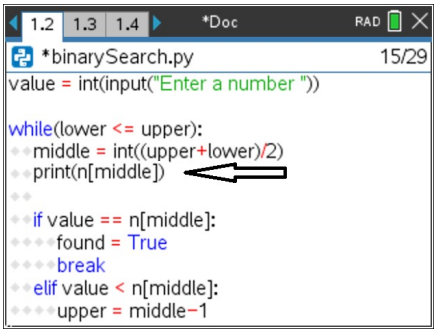
    9
    12
    13
    13
    17
    18
    19
    20
    21
    22
    23
    24
    30

    If the program was to look for value = 16, draw out the checks and remaining list segments after each pass through the binary search. Your result should look similar to the paths in step #7.

18. Enter the list from step 16 into your calculator.

    Add a print(n[middle]) at the top of the loop. After you calculate the value for middle.

    Does your code check/print the numbers you expected in step 17?



19. Let's say you create a list of 300 items.
Roughly, how many items would be checked?

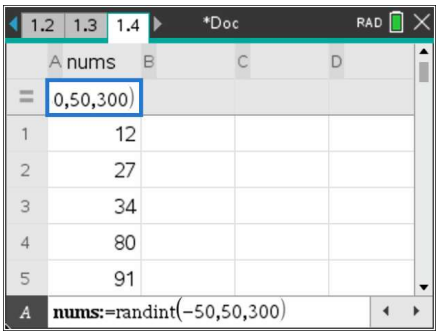
Well, start at the middle of 300

If not found, cut in half and start at the middle of 150 items.

If not found, cut in half and start at the middle of 75 items.
……..

How many total comparisons would it make?

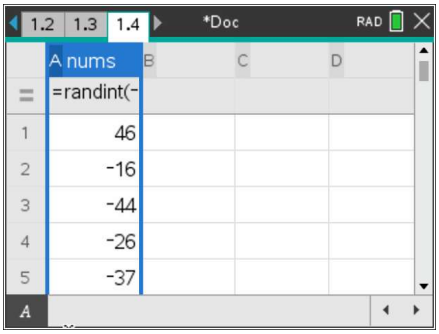
20. You can simulate the problem above using your program.
Go back to your spreadsheet page.



In the formula cell (second row with the = sign), type the formula:
randint(-50,50, 300)
Press enter.

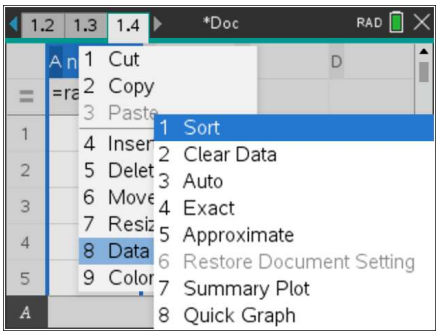
This will generate 300 random integers from -50 to 50. Scrolling through the numbers, you should see 300 random numbers between -50 and 50.

One slight issue, the binary search only works on lists that are in order.
This list is not ordered……..yet.



Click on the label "nums", then press the up arrow to select the entire list.

With the column selected, right click on the row.
Choose Data, Sort.



This could take a few moments, but it should put your list in order.

21. Now that you have 300 sorted items from -50 to 50, run your program.

    Search for 700. It is not in the list, but it will let you see how many comparisons are made since you added the print statement in step #17.

    How many comparisons did it make for 300 items?

22. Complete the following table:
    If you want to use your program to complete the table, remember step 3 shows you how to clear a list.

Number of Items in List	Sequential Search: Maximum Number of Comparisons	Binary Search: Maximum Number of Comparisons
3	**3**	**2**
4		
5		
100		
300		
500		
1000		