Problem #2

Goal

To test whether a recursive, iterative or linked list type binary search is faster by testing it on arrays of sizes 1 million and 10 million filled with random numbers.

Approaches

- Create the 2 arrays in the heap because arrays of that size cannot be created in the stack
- Fill the arrays with random numbers
- Sort both arrays using sort function in algorithm library because a binary search requires a sorted array or list
- Create linked lists from the sorted arrays
    - Creating the linked lists utilized the creation of the Node class, a function to insert a new Node and append it to the end of the list and an arrayToList function that looped the insert function using values from the array passed through the parameter
    - Copying the sorted array values to create a new linked list was used instead of generating random values during the instantiation of the list because there was already a sorted set of ints and it helped verify the functions were working correctly by comparing the results to the recursive and iterative searches and seeing if an index was found or not found as well
- Perform a recursive binary search, timed and displayed

  - the function for recursive search passes the array, an int for the left index, an int for the right index and an int val to represent the value being searched for
  - the base case for the recursion is if the right index is <=1 and if reached then desired val was not found and an value of -1 is returned indicating no index exists
  - find a mid index in the middle of the left and right and compare the value there to the one being searched for
    - if it is equivalent return the int mid which is the index where val is
    - if the value at mid is less than the desired val then it must be somewhere to the right of the mid point if it exists and perform the recursive function again but with the new left range of mid+1 to the same rightmost index
    - if the value at mid is more than the desired val then it must be somewhere to the left of the mid point if it exists and perform the recursive function again but with the new right range of mid-1 to the same leftmost index
- Perform an iterative binary search, timed and displayed
  - the iterative search is similar to the recursive but instead of recursion a while loop is used with the base case of when the left and right indices meet or right is less than the left
  - a mid value is still created to represent the middle index between left and right but rather run a recursive call to the function again it just changes the value of l or r depending on whether the desired val is larger or smaller to the mid

    value respectively in order to loop the program and change the value of mid every iteration
- Perform a binary search on the created linked lists, timed and displayed
    - A middle function was created that returns the middle Node between two Nodes passed in the parameters. This was done by creating two Nodes called fast and slow with slow traversing the list one Node at a time and fast traversing at 2 nodes

<u>Issues</u>
1. Due to the large sizes of 1 million and 10 million the program takes a long time to run
2. The recursive and iterative functions were also much faster compared to the linked list binary search but comparing the two is difficult because during many runs a time of 0 microseconds showed for both making them equivalent.
3. The middle function works by creating 2 Nodes every time it is called to find the desired output and although it is useful and works for any size list, the size is already known so there might be a faster method.

<u>Solutions</u>
1. During the debugging process, smaller values were initially used just to make sure the program runs properly.
2. The results are that the linked list type binary search was much slower due to the structure having slower search times

while the recursive and iterative searches used an array where traversal is faster and is O(log n).

3. middle function used in linkedBinary function may be shortened to not use the fast and slow pointers concept and instead since we know the exact size and have no need to create 2 Nodes every time the function is called as well.

Results

The program found the recursive and iterative searches utilizing an array structure to be much faster than the linked list structure when all three were using a binary search. Even the creation was timed and found arrays to be faster in that regard as well.

Problem #8

Goal

To create a templated class that effectively finds all possibilities of a list of random numbers that add to some s.

Approaches

- get user input for size of array
- fill array with random values and display the values
- get user input for targeted value sum
- call on templated class printAllSubsets with class ints
    - The class has a function that will print all subsets using recursion
    - The base case for the recursion is when the remaining targeted sum is 0 then it will print the subset that was stored

in a vector and when there are no more elements remaining to traverse through
- There are 2 recursion calls because there is a case when an element is included and another when an element is not included
- When the base case fails the program is done and has printed a subset sum on a new line

Issues
1. Could’ve had a better templated class

Solutions
1. The templated class is only the printAllSubsets function and could have had a constructor class and private variables to go along with it.

Results
The program executes a templated class to find all subsets of a randomized array of size determined by the user and displays them using a recursive function.

Problem #9
Goal
Create a structure filled with random 0s and 1s of size t and create a function to check if it is a DeBruijin sequence of B(2,k). If the random array is not a DeBruijin sequence then randomly

mutate each spot until it is. Perform with an array and with a linked list 100 times each, time it and compare results.

<u>Approaches</u>

- Take user input to get degree of 2 for the size of array and linked list
- start the clock for the array structure and time the mutation of 100 arrays
    - inside the for loop, create the arrays on
- start the clock for the linked list structure and time the mutation of 100 linked lists

<u>Issues</u>

1. inefficient mutation where needs assignment every time even when not changing
2. the boolean function to determine whether or not the sequence is a DeBruijin was unable to be finished

<u>Solutions</u>

1. mutate only when the 5% probability is met by passing the array in the parameter through a pointer and changing the value directly rather than passing the value and reassigning it
2. Maybe implement a circular linked list because a DeBruijin sequence can be mapped that way

<u>Results</u>

Although the algorithm for the boolean function to determine whether a sequence is DeBruijin was not completed, the program times and shows that the creation of 100 arrays compared to 100 linked lists was faster. Knowing that the program would traverse through the array and change the values but not change the size of the structure through insertion or deletion would mean that ultimately the array structure would be much faster than the linked list structure.