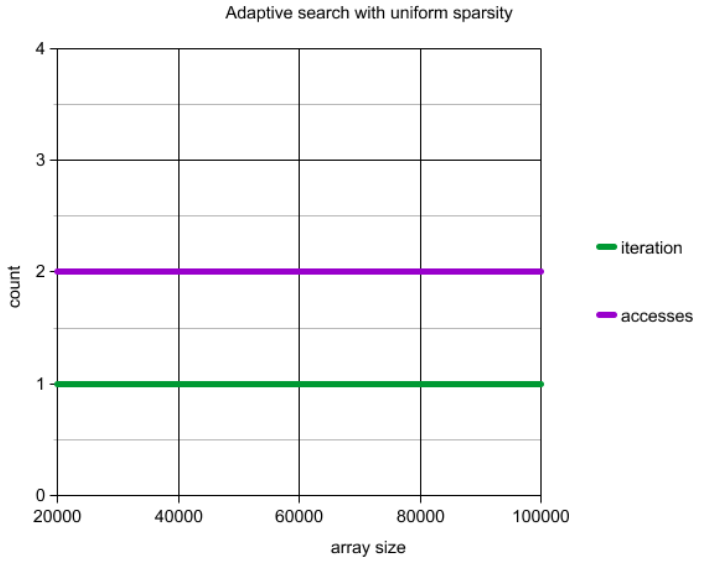
Hyunsoo Im
G01088929
CS 483

Project on Adaptive and Interpolation binary search

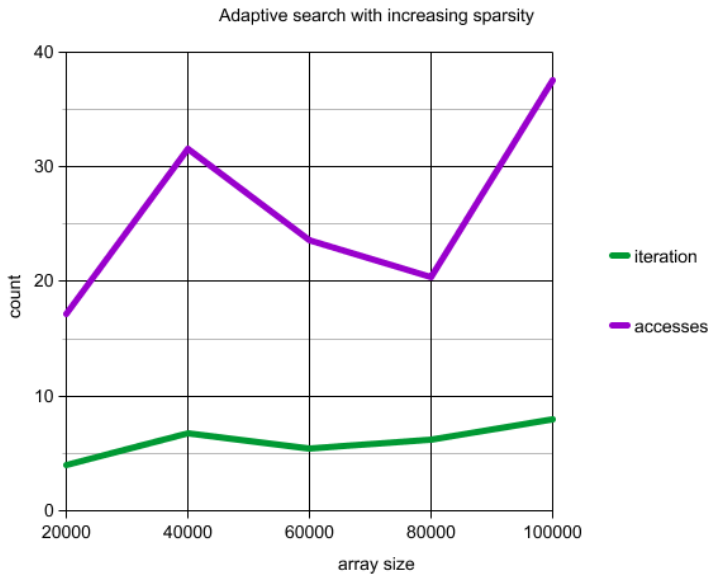
This project's goal is to implement three different algorithms that implement both interpolation and binary search but in a different way. These are, adaptive search, Interpolation-binary search and a third algorithm that runs both interpolation search and binary search then simply exits if one solves faster than other.

Adaptive search. Adaptive search is an algorithm that prioritizes shortening the boundaries between high and low. The way I implemented it in my code is first, it checked which boundary between Interpolation and Binary search was smaller. Then I would update the low and the high based on which was smaller and this way, the program has a smaller range of lists to work with each iteration. However I believe doing this causes a lot more comparisons to occur. In a list of uniform sparsely between each number, it works well as the best case will be 0(1) as interpolation search will usually respond faster than binary search. Graph 1 below shows average accesses and iteration from 2 to 100000 element size arrays with 20000 intervals (This means any list size from 0 - 20000 tested 10 times, then 20000 - 40000 tested 10 times etc). I will be testing each interval 10 times and getting the average. Easy enough, because in a uniform sparsity, adaptive search acts like an interpolation search as it is O(1) to search. However when it's not uniform sparsity and instead, increasing, that is a whole different conversation.



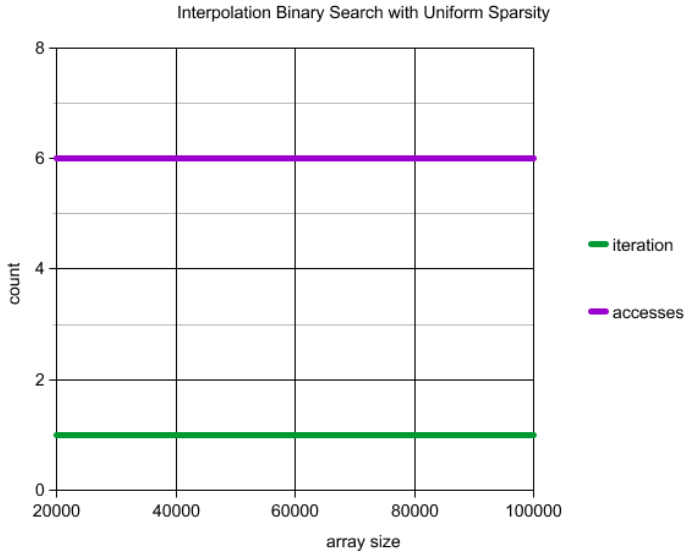
In an increasing sparsity, it is a different conversation because interpolation search is not very reliable when searching in a non-uniform sparsity. This means the numbers in between each element are not the same. My list generator for increasing randomly selects a number from 2 to 100000 elements to be the length of the list. Then selects a random number from 2 to the randomly selected size and adds that value from the previous number added. (first element initialized with 1). Unfortunately, interpolation search in a non-uniform becomes O(n). Adaptive search therefore implemented binary search into their algorithm. This way, when the list is not uniformly sparse, binary search takes over every iteration and gets the complexity of O(logn). Below is the chart for its iteration and accesses under increasing sparsity.

Average accesses through 50 tests (10 in each interval) = 26.08 accesses
Average Iteration through 50 tests = 6.08 iterations



Interpolation Binary Search

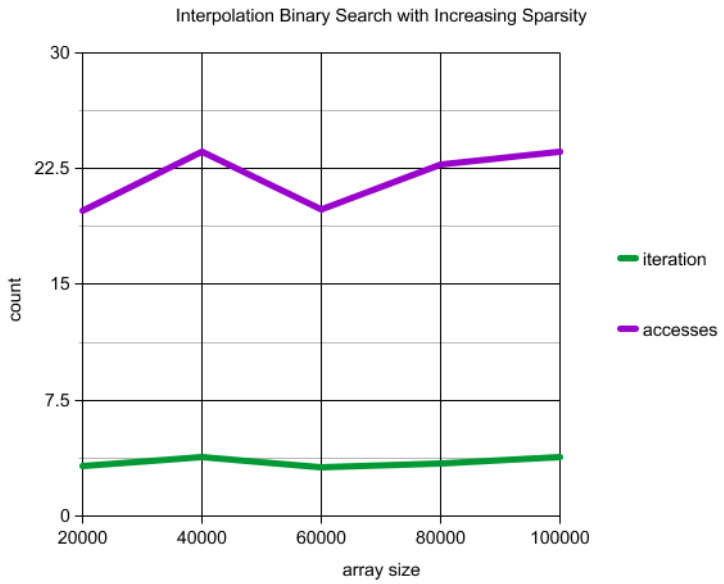
Interpolation Binary search is another algorithm that utilizes both interpolation and binary search but in a different way. It is a research done by Santoro and Sidney and they manipulate the interpolation search that tries to create a smaller boundary. This algorithm also starts with interpolation search then tries binary search after. However they implemented theta and S that are set to be two in order to decrease the interaction to 4/3 [log log n + 1]. In a uniformly sparse list, this algorithm also maintains O(1) complexity for search Below is the chart for a uniformly sparse list.



In an increasingly sparse list, it's different.

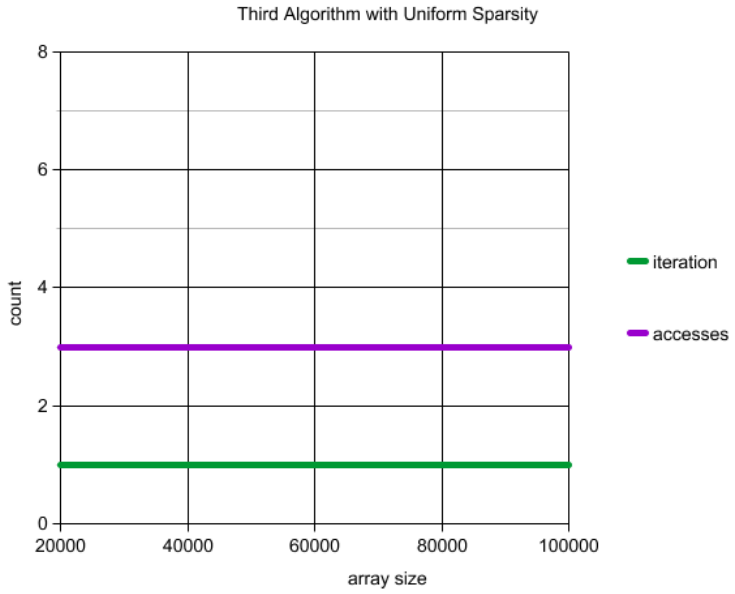
Average accesses through 50 tests = 21.906 accesses

Average Iteration through 50 tests = 3.479 iterations

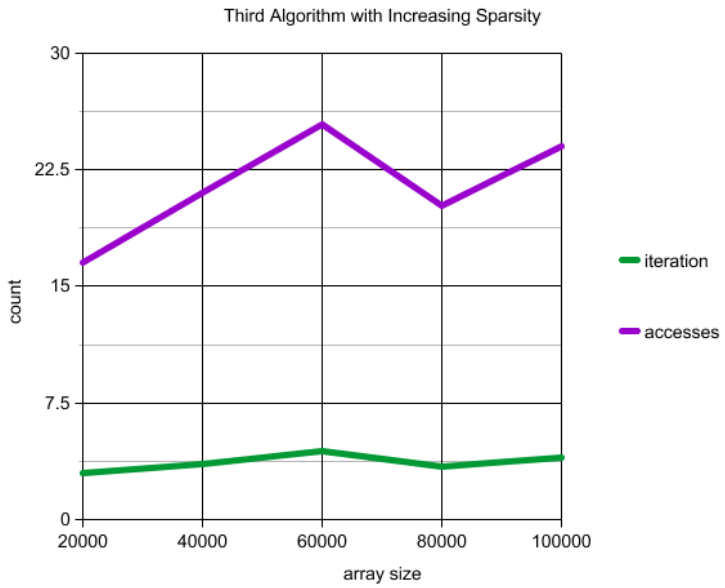


Third Algorithm:

This algorithm is simply running both interpolation and binary search in the same while loop and if one finds the index before the other, it will simply exit. This way it is able to use interpolation search's speedy complexity of O(1) in an uniformly sparse list and binary search's best case of O(logn) in an increasing sparsity list. I implemented this algorithm that simply has 2 individual low and high trackers for both IS and BS (for easier readability and debugging) that simply updates according to its algorithm. Then it exits as soon as one of them finds the answer. Chart below represents this algorithm in an uniformly sparse list.



Below is the chart that represents the increasing sparsity for this third algorithm.
Average accesses through 50 tests = 21.42 accesses
Average Iteration through 50 tests = 3.68



Concluding the testing, it is resulted that all three algorithms had complexities of O(1) to find the key no matter the size in a uniformly sparse list. However in an increasing sparse list, adaptive search seems to take the most iterations as well as the most comparisons. This may be due to focusing too much on shortening the range of each iteration causing a lot more comparisons to occur. And because interpolation search is not very reliable when searching in an increasing list, binary search will almost always take over leading to more iterations. Reviewing the algorithms, it is thought that the third algorithm should have similar run time as adaptive search as is only running IS and BS. However this difference may be due to human error and difference on how I implemented the adaptive search. All in all, I disagree with Bonasera's research paper on his algorithm because it does cause a lot more comparisons than the other two causing unnecessary delays.