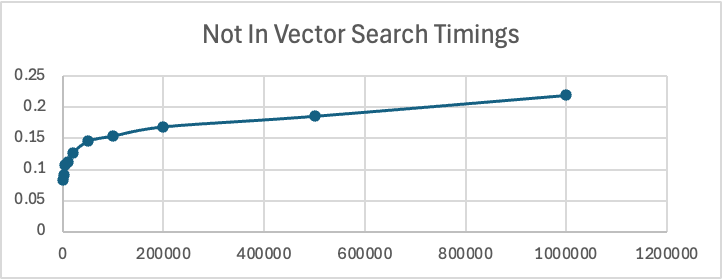
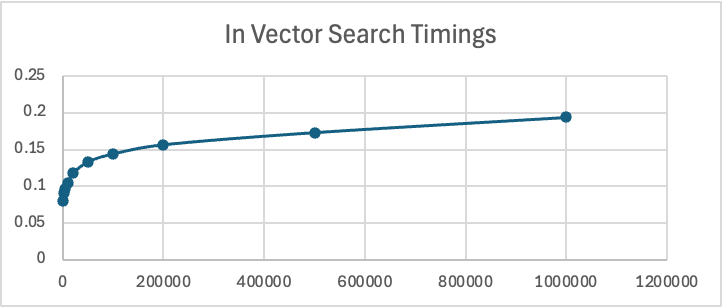
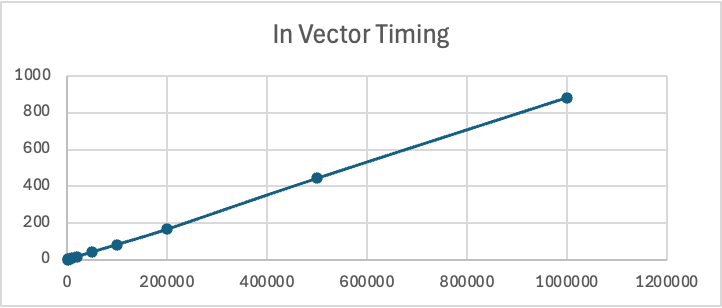
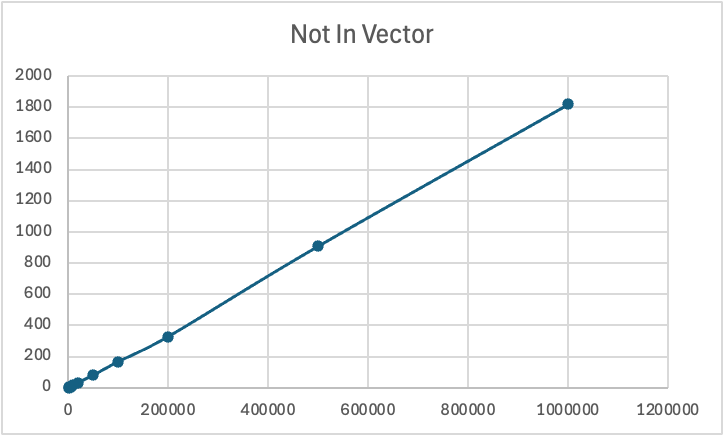
**Lab5a Binary and Sequential Search**

Binary Search Timings and Explanation

Below are the two graphs for the binary search. One is in-vector, and the other is not in-vector. These are shaped as O(lg n) because the algorithm is repeatedly halving the search interval. It compares the target value to the middle element of the array (it must be sorted!) and, if they match, we’re done. If the target is < the middle element, we search the left half. If it’s > the middle element, we search the right. This continues until we find the number or we have searched through the vector finding nothing. 

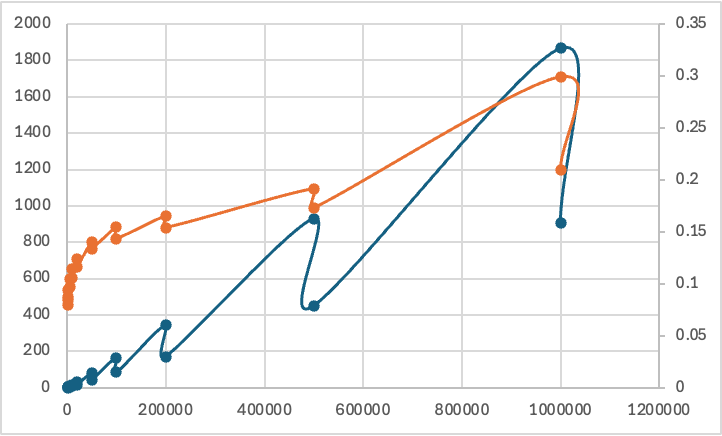
Binary Search Timings and Explanation

Below are the two graphs for the sequential search. These end up as O(n) timings because the search depends on the size of the vector. In the worst-case scenarios, you don’t the number, so the program must search through every element in the vector to check for the target value. That is why you see the not in-vector search timings shoot up even higher than the in-vector timings – it searches the whole vector for every test case. We expect this behavior because we intentionally made it search for a value not in the vector, but it’s still interesting to see the difference in these graphs.  

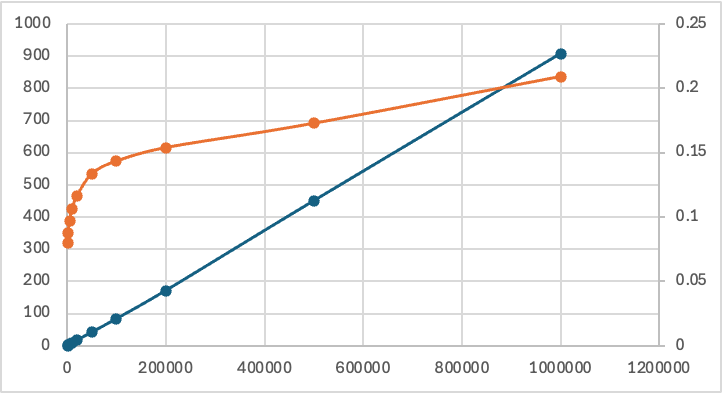
Graphs of Combined Timings for All Test Cases

I made a few different graphs just because I wanted to see what the data looked like in multiple ways. I’m not sure if I’m seeing what you intended with the combination of the graphs. Even when I separate them out below, they still look a little funky. I think I’ll ask you about this in class.

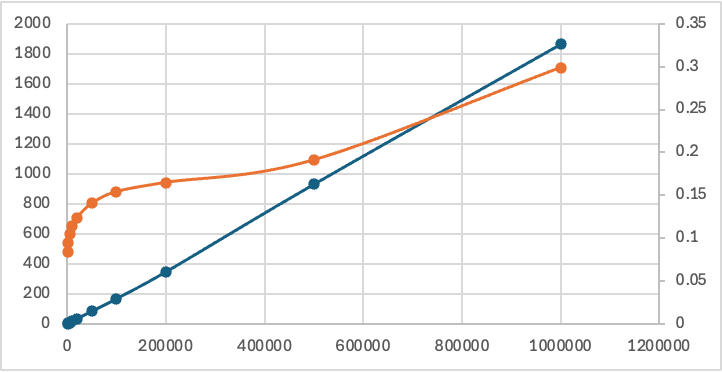
Combined:



In-Vector:



Not In-Vector:



Code Appendix:

binSearch.cpp:

1. #include <iostream>

2. #include <vector>

3. #include <chrono>

4. #include <fstream>

5. #include <algorithm>

6. #include <iomanip>

7. using namespace std;

8.

9. bool binarySearch(const vector<unsigned long long>& alist, unsigned long long item){

10. int first = 0;

11. int last = alist.size() - 1;

12. bool found = false;

13. while (first <= last && !found){

14. int midpoint = (first + last) / 2;

15. if (alist[midpoint] == item) {

16. found = true;

17. } else {

18. if (item < alist[midpoint]) {

19. last = midpoint -1;

20. } else{

21. first = midpoint +1;

22. }

23. }

24. }

25. return found;

26. }

27.

28. unsigned long long getInt(int numDigits = 10) {

29. unsigned long long num = 0;

30.

31. num = rand() % 9 + 1;

32. for (int i = 0; i < numDigits -1; i++) {

33. num = num \* 10 + rand() % 10;

34. }

35. return num;

36. }

37.

38. void getUniqueNums(int sz, vector<unsigned long long>& v) {

39. vector<unsigned long long> tmpv;

40. vector<bool> bvect;

41. bvect.resize(10000000000, false); // resizing vector large enough to track 9-digit #s

42.

43. int ix = 0;

44. while(ix < sz \* 1.1 && v.size() < sz) {

45. unsigned long long num = getInt(9);

46. if (bvect[num] == false) {

47. v.push\_back(num);

48. bvect[num] = true;

49. }

50. ix++;

51. }

52. bvect.resize(0);

53.

54. cout << v.size() << " unique numbers generated!" << endl;

55. }

56.

57. int main() {

58. ofstream csvFile("bin\_timing.csv");

59. csvFile << "Size,SearchType,Time(milliseconds)" << endl;

60.

61. cout << "Generating Timing Data..." << endl;

62.

63. vector<int> testSize = {1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000, 1000000};

64. int count = 10;

65.

66. for (int size : testSize) {

67. vector<unsigned long long> vec;

68. getUniqueNums(size + 1000, vec); // This is where we generate 1000 extra ints

69.

70. // Copying those unique nums in a new vector except for the last 1000

71. vector<unsigned long long> uniqueVec(vec.begin(), vec.end() - 1000);

72.

73. // Quick little sort function

74. sort(uniqueVec.begin(), uniqueVec.end());

75.

76. // First timing where we're searching for integers NOT in the vector

77. vector<unsigned long long> notInVec(vec.end() - 1000, vec.end());

78.

79. auto start1 = chrono::high\_resolution\_clock::now();

80. for (unsigned long long num : notInVec) {

81. bool found = binarySearch(uniqueVec, num);

82. }

83. auto end1 = chrono::high\_resolution\_clock::now();

84. chrono::duration<double, std::milli> totalNotInVec = end1 - start1;

85.

86. // Second timing where we're searching for integer IN the vector

87. vector<unsigned long long> inVec;

88. int selectNum = min(1000, (int)uniqueVec.size());

89. for (int i = 0; i< selectNum; i++) {

90. int randIndex = rand() % uniqueVec.size();

91. inVec.push\_back(uniqueVec[randIndex]);

92. }

93.

94. auto start2 = chrono::high\_resolution\_clock::now();

95. for (unsigned long long num : inVec) {

96. bool found = binarySearch(uniqueVec, num);

97. }

98. auto end2 = chrono::high\_resolution\_clock::now();

99. chrono::duration<double, std::milli> totalInVec = end2 - start2;

100.

101. csvFile << size << ",NotInVec," << fixed << setprecision(4) << totalNotInVec.count() << endl;

102. csvFile << size << ",InVec," << fixed << setprecision(4) << totalInVec.count() << endl;

103. }

104. csvFile.close();

105. return 0;

106. }

107.

seqSearch.cpp:

1. #include <iostream>

2. #include <vector>

3. #include <algorithm>

4. #include <fstream>

5. #include <chrono>

6. #include <iomanip>

7.

8. using namespace std;

9.

10. bool sequentialSearch(const vector<unsigned long long>& alist, unsigned long long item){

11. int pos = 0;

12. bool found = false;

13.

14. while (pos < alist.size() && !found) {

15. if (alist[pos] == item) {

16. found = true;

17. } else {

18. pos = pos+1;

19. }

20. }

21. return found;

22.

23. }

24.

25. unsigned long long getInt(int numDigits = 10) {

26. unsigned long long num = 0;

27.

28. num = rand() % 9 + 1;

29. for (int i = 0; i < numDigits -1; i++) {

30. num = num \* 10 + rand() % 10;

31. }

32. return num;

33. }

34.

35. void getUniqueNums(int sz, vector<unsigned long long>& v) {

36. vector<unsigned long long> tmpv;

37. vector<bool> bvect;

38. bvect.resize(10000000000, false); // resizing vector large enough to track 9-digit #s

39.

40. int ix = 0;

41. while(ix < sz \* 1.1 && v.size() < sz) {

42. unsigned long long num = getInt(9);

43. if (bvect[num] == false) {

44. v.push\_back(num);

45. bvect[num] = true;

46. }

47. ix++;

48. }

49. bvect.resize(0);

50.

51. cout << v.size() << " unique numbers generated!" << endl;

52. }

53.

54. int main() {

55. ofstream csvFile("seq\_timing.csv");

56. csvFile << "Size,SearchType,Time(milliseconds)" << endl;

57.

58. cout << "Generating Timing Data..." << endl;

59.

60. vector<int> testSize = {1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000, 1000000};

61. int count = 10;

62.

63. for (int size : testSize) {

64. vector<unsigned long long> vec;

65. getUniqueNums(size + 1000, vec); // This is where we generate 1000 extra ints

66.

67. // Copying those unique nums in a new vector except for the last 1000

68. vector<unsigned long long> uniqueVec(vec.begin(), vec.end() - 1000);

69.

70. // First timing where we're searching for integers NOT in the vector

71. vector<unsigned long long> notInVec(vec.end() - 1000, vec.end());

72.

73. auto start1 = chrono::high\_resolution\_clock::now();

74. for (unsigned long long num : notInVec) {

75. bool found = sequentialSearch(uniqueVec, num);

76. }

77. auto end1 = chrono::high\_resolution\_clock::now();

78. chrono::duration<double, std::milli> totalNotInVec = end1 - start1;

79.

80. // Second timing where we're searching for integer IN the vector

81. vector<unsigned long long> inVec;

82. int selectNum = min(1000, (int)uniqueVec.size());

83. for (int i = 0; i< selectNum; i++) {

84. int randIndex = rand() % uniqueVec.size();

85. inVec.push\_back(uniqueVec[randIndex]);

86. }

87.

88. auto start2 = chrono::high\_resolution\_clock::now();

89. for (unsigned long long num : inVec) {

90. bool found = sequentialSearch(uniqueVec, num);

91. }

92. auto end2 = chrono::high\_resolution\_clock::now();

93. chrono::duration<double, std::milli> totalInVec = end2 - start2;

94.

95. csvFile << size << ",NotInVec," << fixed << setprecision(4) << totalNotInVec.count() << endl;

96. csvFile << size << ",InVec," << fixed << setprecision(4) << totalInVec.count() << endl;

97. }

98. csvFile.close();

99. return 0;

100. }

101.