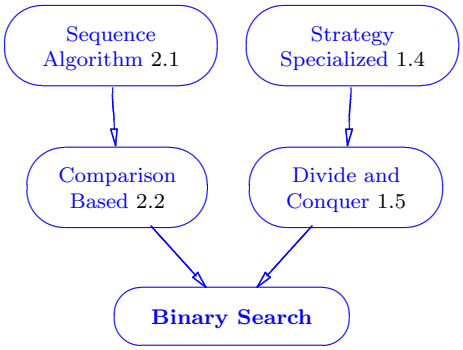
55 Binary Search

Section authors: Matthew Goodyear, Rachel Vecchitto, Matthew Zuckerman



Refinement of: Comparison Based (§2.2), therefore of Sequence Algorithm (§2.1), and Divide and Conquer (§1.5), therefore of Strategy Specialized (§1.4).

Prototype:

```
template⟨class ForwardIterator,        class LessThanComparable⟩
bool binary_search(
     ForwardIterator first,
     ForwardIterator last,
     const LessThanComparable& value);

template⟨class ForwardIterator,        class T,        class StrictWeakOrdering⟩
bool binary_search(
     ForwardIterator first,
     ForwardIterator last,
     const T& value,
     StrcitWeakOrdering comp);
```

Description: This implementation of binary search returns true if an element equivalent to value is present in [first,last), and false otherwise. The first version of binary search uses the less than operator for comparison, while the second uses the function object comp.

Asymptotic complexity: Let $N = \text{last} - \text{first}$.

- Average case: $O(\log N)$
- Worst case: $O(\log N) + 2$

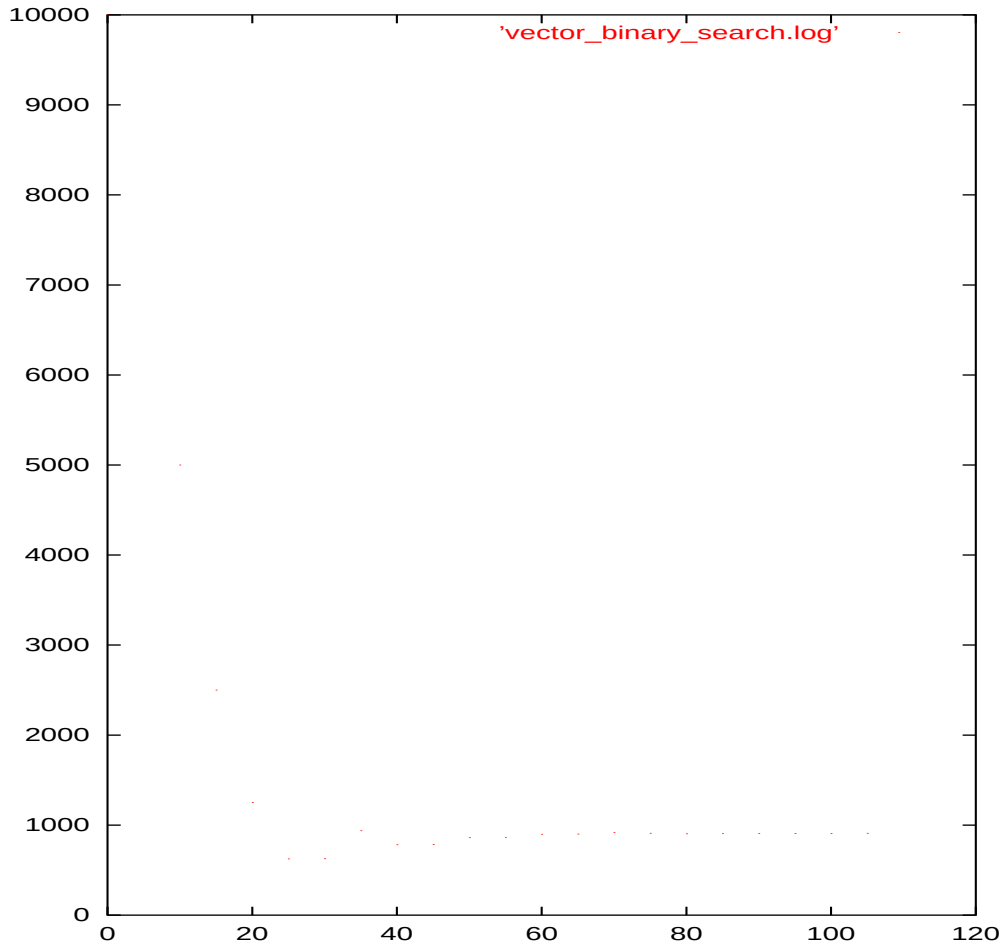
Complexity in terms of operation counts:

- Average case using a bidirectional iterator:
  Comparisons: $1/2 \log_2 N + 6.244$
  Assignments: 1
  Iterator ops: $10^5 * (0.094 * \log_2 N - 1.067)$
  Integer ops: $10^5 * (0.22 * \log_2 N - 2.49)$
- Average case using a random access iterator:
  Comparisons: $1/2 * \log_2 N + 6.244$
  Assignments: 1
  Iterator ops: $1.991 * \log_2 N + 47.114$
  Integer ops: $7.481 * \log_2 N + 56.89$
- The following table provides operation counts for runs of binary search using both bidirectional and random access iterators:

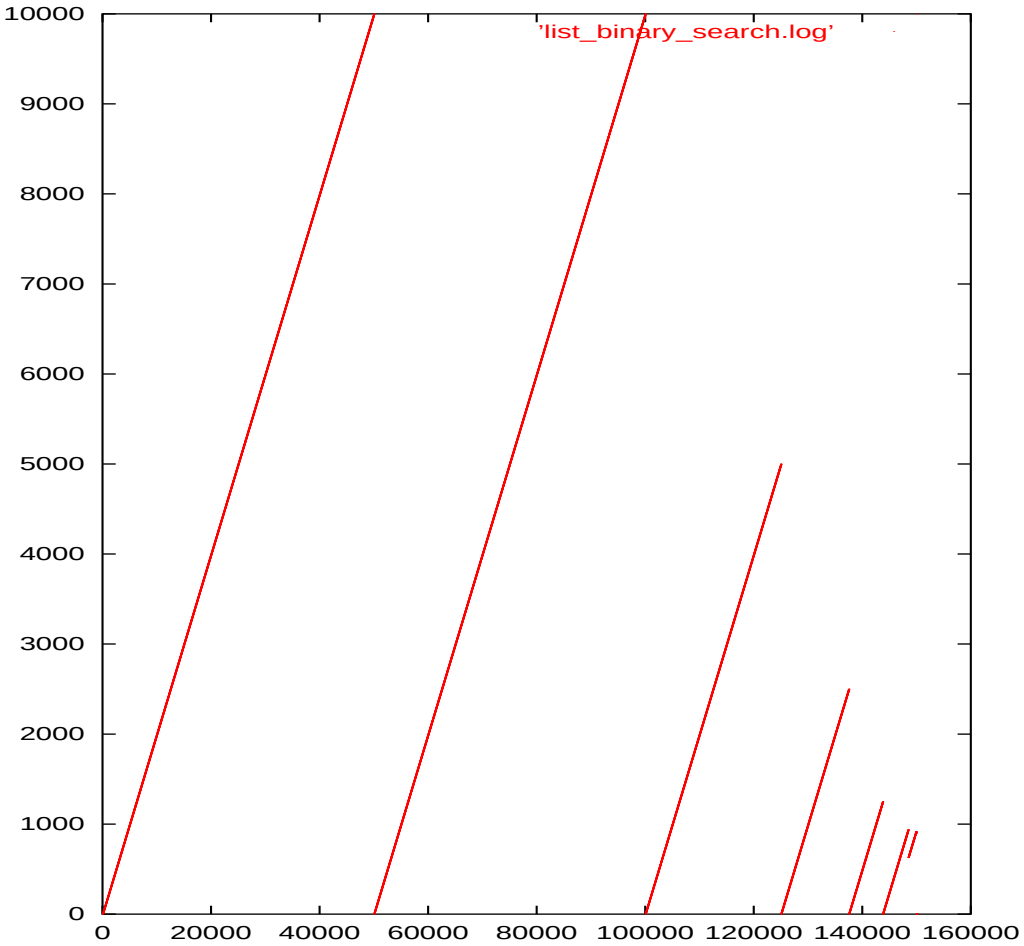
Table 1: Performance of Binary Search Using Various Input Sizes and Iterator Types

Size	Iterator Type	Comparisons	Assignments	Iterator Ops	Integer Ops	Total Ops
1000	Bidir.	11	1	3050	7134	10196
	Random	11	1	67	127	206
2000	Bidir.	12	1	6053	14143	20209
	Random	12	1	74	149	236
4000	Bidir.	12	1	6048	14132	20193
	Random	12	1	67	139	219
8000	Bidir.	13	1	12048	28149	40211
	Random	13	1	72	153	239
16000	Bidir.	13	1	12048	28149	40211
	Random	13	1	72	153	239
32000	Bidir.	14	1	24056	56181	80252
	Random	14	1	78	173	266
64000	Bidir.	14	1	24056	56181	80252
	Random	14	1	78	173	266
128000	Bidir.	15	1	48060	112186	160262
	Random	15	1	83	190	289
256000	Bidir.	15	1	48060	112186	160262
	Random	15	1	83	112186	160262
512000	Bidir.	16	1	96060	224205	320282
	Random	16	1	86	200	303
1024000	Bidir.	16	1	96060	224205	320282
	Random	16	1	86	200	303

Iterator trace plot:



An ordered vector of ten thousand elements is being searched for a single value by binary search. Random Access Iterators are being used to find the middle element of the vector, at which time a comparison is made. If the value being searched for is greater than the value of the middle element, then this process is recursively repeated on the greater half of the vector. If the value being searched for is less than the value of the middle element, then this process is recursively repeated on the lesser half of the vector. The process will terminate when either an element with a value matching that being searched for is found, or when the recursion results in the search of a single element with a value not equal to the value being searched for.



The same process is used to search an ordered list of elements for a single value as that used on the vector. The difference is that the list container does not provide random access to its elements. The list container does provide Bidirectional Iterators, which are used to traverse the elements. As in the case of the binary search on the vector, comparisons of the value of an element versus that of the value being searched for are only performed on the middle element of the section being recursively searched.

Animation:

- A demonstration of binary search.