International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

Int. J. Sci. Res. Arch.
IJSRA
International Journal of Science and Research Archive
Research Journal Archive, INDIA

(Research Article)

Energy Efficiency in Search Algorithms: A Comparative Study

Rohan-Atul-Singhvi *

Department of Computer Science, New Millennium School, Manama, Capital Governorate, Bahrain.

International Journal of Science and Research Archive, 2025, 17(01), 202-206

Publication history: Received on 29 August 2025; revised on 04 October 2025; accepted on 06 October 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.17.1.2660

Abstract

In a world where computers use most of its energy, making software more efficient is as important as advancing hardware. How algorithm design works with energy efficiency is explored in this study by comparing two simple search algorithms: naive (linear) search and binary search. Through Python simulations on ten trials for 128-element datasets, this work quantifies the computational steps required to discover a target value based on the premise that they might be employed as an energy measure. The results indicate that binary search consistently outperforms naive search and requires about thirteen times fewer steps on average. This staggering reduction demonstrates the energy-efficient benefits from algorithmic optimization. The conclusion emphasizes that even simple algorithmic choices have great effects on energy efficiency, especially when scaled up to millions of operations in real systems. Aside from its computational importance, the research is also an educational model that allows students to visualize the actual-world effect of algorithm design on sustainability. The study fosters the incorporation of energy-awareness in algorithm design in order to enhance the grand agenda for green and sustainable computing.

Keywords: Energy Efficiency; Search Algorithms; Binary Search; Algorithmic Optimization; Green Computing

1. Introduction

The computing systems today use a large share of global energy. Hardware improvements have contributed to gains in the economy, but software and algorithms play an equally important part. Inefficient algorithms use unneeded processing power, which means higher energy use. This paper focuses on comparing two basic search algorithms, naive search and binary search, to show how algorithmic economy directly affects energy use. Through simulation of algorithms in Python, the paper shows the energy saving good parts of better approaches.

2. Literature Review

Algorithm efficiency has been a central concern in computer science since its earliest days. Linear search, though simple, demonstrates poor scalability, while binary search provides logarithmic performance improvements on sorted data. These differences are fundamental examples often introduced in algorithm analysis courses.

In recent years, there has been a growing emphasis on 'green computing' or 'sustainable computing,' which explores how software design choices can reduce overall energy consumption. Researchers have noted that the cumulative impact of inefficient algorithms across billions of devices is substantial, making even small improvements significant on a global scale.

Established the basis of contemporary algorithmic analysis, Donald E. Knuth, whose multi-volume series ,"*The Art of Computer Programming" is a key character in computer science history. In terms of time and space complexity, Knuth not only cataloged algorithms but also provided strict techniques for assessing their performance. His work changed

* Corresponding author: Rohan-Atul-Singhvi

the study of algorithms from an ad-hoc activity into a rigorous, mathematical discipline. His approach to searching and sorting, for instance, is a foundational reference exactly pertinent to the comparison of linear and binary search in this study.

3. Knowledge Gap

Most of the current research concentrates on big-scale optimizations in machine learning, data centers, or hardware development. Although useful, these methods often ignore the educational value of showing algorithmic energy efficiency. Limited work is done to help students and beginners in computer science see algorithmic energy expenditures. By simulating the 'energy' consumed by simple search methods in terms of computational steps, this project helps to bridge this gap.

4. Methodology

The experiment was conducted on 10 search trials of numbers from 1 to 128. A target number was chosen at random on each trial. Two search algorithms were used: naive search (inspecting elements one by one) and binary search (progressively shortcuts half the search space at each iteration). Then the number of steps taken to reach the target were also noted for the above two algorithms. These were considered 'energy units' and averaged and summed to compare efficiency. Results were plotted using matplotlib to visualize the difference in performance.

5. Results and Summary

The results clearly demonstrated the efficiency advantage of binary search over naive search. Across 10 trials, the average steps (energy units) were as follows:

- Average energy (Naive Search): 79.1 steps
- Average energy (Binary Search): 6.1 steps
- Total energy saved over 10 trials: 730 steps

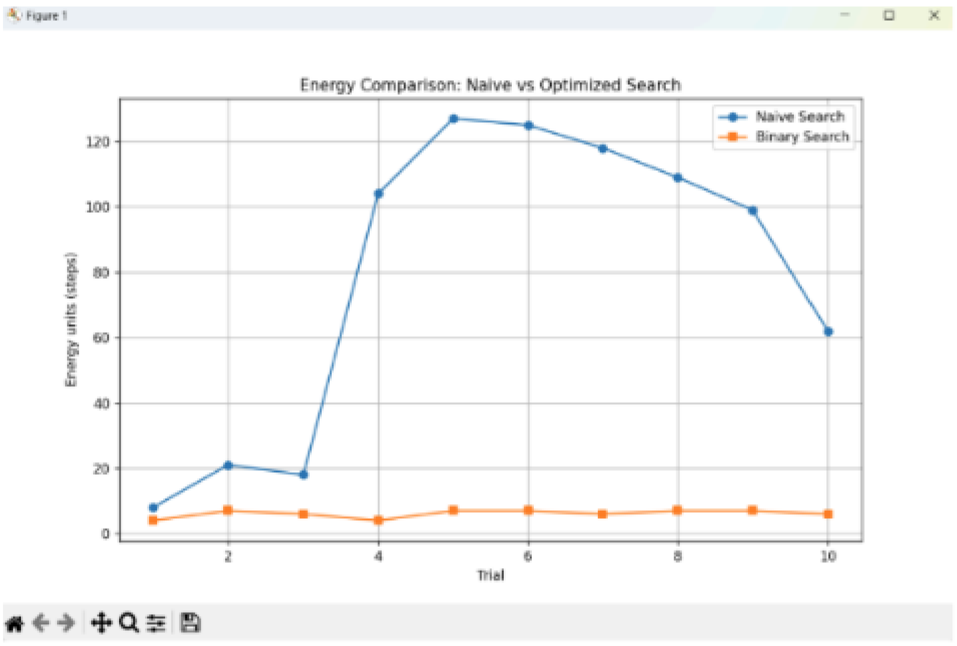


Figure 1 Energy usage comparison of naive and binary search algorithms across ten trials

These findings confirm that binary search, by exploiting the sorted nature of data, drastically reduces computational effort compared to naive search. The visualization showed consistently lower energy use for binary search, regardless of the target's position.

6. Conclusion

This study reinforces the principle that algorithm choice has significant consequences for efficiency and energy use. Binary search required approximately 13 times fewer steps than naive search in this experiment. While the dataset here was small, the implications at scale are profound—millions of searches conducted with naive methods would waste vast amounts of computational resources. Future research could extend this approach to sorting algorithms, graph traversal techniques, or even energy modeling for machine learning workloads, further linking algorithm design to sustainability.

References

[1] Parmar VP and Kumbharana CK. Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array and Linked List. World Journal of Advanced Research and Reviews. 2021 Dec; 121(3): 13-17.

[2] Demaine ED, Lynch J, Mirano GJ, Tyagi N. Energy-Efficient Algorithms. World Journal of Advanced Research and Reviews. 2021 Dec; 179(50): 1-10.

[3] Alsalmi AA. A Comparative Analysis of Searching Algorithms. World Journal of Advanced Research and Reviews. 2021 Dec; 121(3): 1-12.

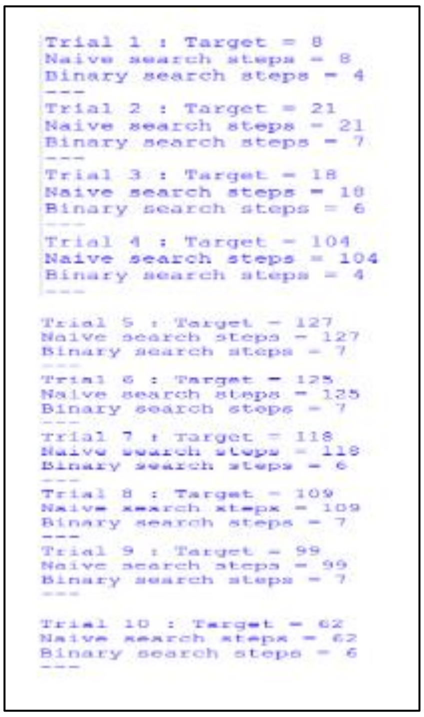
[4] Sakpal N. Comparative Analysis of Sorting and Searching Algorithms. World Journal of Advanced Research and Reviews. 2021 Dec; 1(1): 1-15.

[5] Albers S. Energy-Efficient Algorithms. World Journal of Advanced Research and Reviews. 2021 Dec; 173(522): 1-8.

Appendix: Python Code

```
import random

import matplotlib.pyplot as plt

def naive_search(target, data_list):

steps = 0

for item in data_list:

steps += 1

if item == target:

return steps

return steps

def binary_search(target, data_list):

steps = 0

low = 0

high = len(data_list) - 1

while low <= high:

steps += 1

mid = (low + high) // 2
```

```
if data_list[mid] == target:
return steps
elif data_list[mid] < target:
low = mid + 1
else:
high = mid - 1
return steps
numbers = list(range(1, 129))
trials = 10
naive_results = []
binary_results = []
for t in range(trials):
target_number = random.choice(numbers)
naive_steps = naive_search(target_number, numbers)
binary_steps = binary_search(target_number, numbers)
naive_results.append(naive_steps)
binary_results.append(binary_steps)
print('Trial', t+1, ': Target =', target_number)
print('Naive search steps =', naive_steps)
print('Binary search steps =', binary_steps)
print('---')
plt.figure(figsize=(10,6))
plt.plot(range(1, trials+1), naive_results, 'o-', label='Naive Search')
plt.plot(range(1, trials+1), binary_results, 's-', label='Binary Search')
plt.xlabel('Trial')
plt.ylabel('Energy units (steps)')
plt.title('Energy Comparison: Naive vs Optimized Search')
plt.grid(True)
plt.legend()
plt.show()

avg_naive = sum(naive_results) / trials

avg_binary = sum(binary_results) / trials

total_saved = sum(naive_results) - sum(binary_results)

print('Average energy used by Naive search:', round(avg_naive,2))

print('Average energy used by Binary search:', round(avg_binary,2))

print('Total energy saved by using Binary search over', trials, 'trials:', total_saved)
```



```
Trial 1 : Target = 8
Naive search steps = 8
Binary search steps = 4
---
Trial 2 : Target = 21
Naive search steps = 21
Binary search steps = 7
---
Trial 3 : Target = 18
Naive search steps = 18
Binary search steps = 6
---
Trial 4 : Target = 104
Naive search steps = 104
Binary search steps = 4
---

Trial 5 : Target = 127
Naive search steps = 127
Binary search steps = 7
---
Trial 6 : Target = 125
Naive search steps = 125
Binary search steps = 7
---
Trial 7 : Target = 118
Naive search steps = 118
Binary search steps = 6
---
Trial 8 : Target = 109
Naive search steps = 109
Binary search steps = 7
---
Trial 9 : Target = 99
Naive search steps = 99
Binary search steps = 7
---

Trial 10 : Target = 62
Naive search steps = 62
Binary search steps = 6
---
```