Collin Bauer

# CSCI 310 - Advanced Algorithms

Assignment 2

---

*Chapter 3*

## 3.1

### 2

a) What is the time efficiency of the brute-force algorithm for computing $a^n$ as a function of $n$? As a function of the number of bits in the binary representation of $n$?

> The common brute force algorithm for calculating $a^n$ is to multiply $a$ to itself $n$ times, in other words:
>
> - $a^n = \Pi_{i = 0 \rightarrow n} (a)$
>
> This gives a consistent time complexity of $\Theta(n)$.
>
> The storage complexity is not as simple, as the number of bits needed to store the resulting number is potentially doubled each time $a$ is multiplied by itself. In the worst case, this is $O(a^n)$, which is potentially huge.

b) If you are to compute $a^n$ mod $m$ where $a > 1$ and $n$ is a large positive integer, how would you circumvent the problem of a very large magnitude of $a^n$?

> Since modulus is involved, the largest resulting number will always be <= $m$. This means there is a way we can limit the number of bits used while still getting an accurate answer, saving on storage space.

### 8

Sort the list {E, X, A, M, P, L, E} in alphabetical order by selection sort.

> 0. Start with original list {E, X, A, M, P, L, E}
> 1. Set minimum index to 0
> 2. Compare E to X. E comes first
> 3. Compare E to A. A comes first (swap)
>    {A, X, E, M, P, L, E}
>    *Note: A smarter selection sort might know A is the smallest possible value and move the minimum index right here.*
> 4. Compare A to M. A comes first
> 5. Compare A to P. A comes first
> 6. Compare A to L. A comes first
> 7. Compare A to E. A comes first
> 8. Set minimum index to 1

> 9. Compare X to E. E comes first (swap)
>    {A, E, X, M, P, L, E}
> 10. Compare E to M. E comes first
> 11. Compare E to P. E comes first
> 12. Compare E to L. E comes first
> 13. Compare E to E. Values are equal (no swap necessary
> 14. Set minimum index to 2
> 15. Compare X to M. M comes first (swap)
>    {A, E, M, X, P, L, E}
> 16. Compare M to P. M comes first
> 17. Compare M to L. L comes first (swap)
>    {A, E, L, X, P, M, E}
> 18. Compare L to E. E comes first (swap)
>    {A, E, E, X, P, M, L}
> 19. Set minimum index to 3
> 20. Compare X to P. P comes first (swap)
>    {A, E, E, P, X, M, L}
> 21. Compare P to M. M comes first (swap)
>    {A, E, E, M, X, P, L}
> 22. Comapre M to L. L comes first (swap)
>    {A, E, E, L, X, P, M}
> 23. Set minimum index to 4
> 24. Compare X to P. P comes first (swap)
>    {A, E, E, L, P, X, M}
> 25. Compare P to M. M comes firset (swap)
>    {A, E, E, L, M, X, P}
> 26. Set minimum index to 5
> 27. Compare X to P. P comes first (swap)
>    {A, E, E, L, M, P, X}
>    Done.

### 9

Is selection sort stable?

> Yes. Selection sufficiently compares values such that no two unique values could end up in different locations in the array, therefore, it is stable.

### 11

Sort the list {E, X, A, M, P, L, E} in alphabetical order by bubble sort.

> This one could take upwards of 40+ steps, so I'm skipping this for now...

## 3.2

### 1

Find the number of comparisons made by the sentinel version of sequential search

> A sentinel search is the same as a normal linear search, except it appends the value to search for at the end of the list, and skips checking if the search leaves the bounds of the array. As such, it makes upto *n + 1* comparisons, where *n* is the length of the list. The +1 comes from checking the "sentinel" value.

4

5

How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?

> - 00001 - all one thousand
> - 10000 - 996 (it knows not to check the last four)
> - 01010 - 999 (it knows not to check the last one)

## 3.4

1(a)

Assuming that each tour can be generated in constant time, what will be the efficiency class of the exhaustive-search algorithm outlined in the text for the traveling salesman problem?

> Ane exhaustive travelling saleman search, checking all connections between all nodes, would be $O(n^n)$.

1(b)

## 3.5

6(a)

Explain how one can check a graph's acyclicity by using breadth-first search.

> A breath-first search checks every node for children, then performs another BFS on each child node. This can be used to check if any nodes share a child, as the node would appear more than once in a graph, and therefore, show a cycle in the graph.

6(b)

Does either of the two traversals—DFS or BFS—always find a cycle faster than the other? If you answer yes, indicate which of them is better and explain why it is the case; if you answer no, give two examples supporting your answer.

> This ultimately depends on both the structure of the graph, and what node is chosen as the starting point. I once implemented a DFS on a graph with cycles and noticed that collisions were not obvious.

> My theory is that a BFS is more likely to find these collisions faster, since it visits every child of a node right away, and thus covers more unique nodes in less time.

8(a)

8(b)

---

*Chapter 4*

## 4.1

4

6

11

Let A[0..n − 1] be an array of n sortable elements. (For simplicity, you may assume that all the elements are distinct.) A pair (A[i], A[j]) is called an inversion if i<j and A[i] > A[j].

a) What arrays of size n have the largest number of inversions and what is this number? Answer the same questions for the smallest number of inversions.

> Any array that is in reverse sorted order will have the maximum number of inversion, because with any two indicies i and j where i < j, A[i] will always be greater than A[j].

b) Show that the average-case number of key comparisons in insertion sort is given by the formula

- $C_{avg}(n) \approx n^2/4$

## 4.2

1

> a) node d has no inward edges, so this must be the starting point.
>
> - search according to alphabetical order
> - {d→a, a→b, b→e, b→g, g→f, a→c}
>
> b) node f has no inward edges, so this must be the starting point.
>
> - search according to alphabetical order
> - {f→b, b→c, c→d, d→g, g→e, e→a}

2(a)

5

9

## 4.4

3

a) What is the largest number of key comparisons made by binary search in searching for a key in the following array?

- {3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 93, 98}

> $\log_2 13 = 3.7 \approx 4$ comparisons

10

## 4.5

6

7