CS 3510: Algorithms

Assignment 2

Assignment

Problems identified by x.y(z) denote the problem "y", in chapter "x" of the textbook, with part "z". If "z" is not noted, then the entire problem is required.

Assignment 2a

- 2.5(a, c, e) Use the master theorem, show work.
- Solve recurrence relation T(n) = 2 T(n/3) + n. Use the master theorem, show work.

Assignment 2b

- 2.5(b, d) Use the master theorem, show comparison.
- Solve recurrence relation T(n) = 8 T(n/3) + n^2. Use the master theorem, show work.
- 2.5(g) Use the substitution method. Show the pattern and determination of k_max.
- Complete the tasks for Programming Assignment `binary_search`.

Assignment 2c

- 2.5(f, h) Use the substitution method. Show the pattern and determination of k_max.
- 2.16 Find an algorithm, give pseudo-code, argue correctness, analyze the runtime, showing it is O(log(n)). The values stored are integers, *not necessarily positive* Hint: You should know how to find items in a sorted array in O(log(n)).
- Complete the tasks for Programming Assignment `ternary_search`.

Assignment 2d

- 2.5(i, j) Use the substitution method. Show the pattern and determination of k_max.
- 2.19 Analyze the complexity of the algorithm for part (a). Provide your divide and conquer solution and its complexity analysis for part (b).
- Complete the tasks for Programming Assignment Data Collection.

Assignment 2e

- 2.5(k) Use the substitution method. Show the pattern and determination of k_max.
- 2.22 Find an algorithm, give pseudo-code, argue correctness, analyze the runtime.
- If one algorithm is O(log(m+n)), another is O(log(m) + log(n)), which is more efficient? Give your proof.
- Complete the tasks for Programming Assignment Chart Data.

Assignment 2f

- 2.14 Find a divide-and-conquer algorithm, write the recurrence relation, solve it.
- 2.34 Find a divide-and-conquer algorithm, write the recurrence relation, solve it. The book says "linear". We are not as optimistic. Any polynomial divide-and-conquer algorithm is acceptable.

Assignment 2z, Due Never (optional)

- 2.4(A) Write down the recurrence relation. Solve it.
- 2.4(B) Write down the recurrence relation. Solve it.
- 2.4(part C) Write down the recurrence relation. Solve it.
- 2.4 Which would you choose?
- 2.25(a) Fill in the missing code, give a recurrence relation, and solve it.
- 2.25(b) Fill in the missing code, give a recurrence relation, and solve it.
- 2.17 Find an algorithm, prove the runtime is O(log(n)).

Programming Assignment `binary_search`

- Create a directory in your repository name `02-search` to store your work for this task.
- Use the file `search.cpp` for this task.
- Write the function `unsigned int binary_search( const std::vector< int > &data, int value )`.
- Verify that the function will correctly find the index of `value` within `data`.
- You may assume that `value` is present, and `data` is already sorted in ascending order.
- At the top of your source file, include a comment with your estimated Big-Oh complexity of the algorithm.

- In the first pass of your code, write it to handle vectors whose sizes are powers of 2.
- In the second pass of your code, write it to handle vectors whose sizes are not powers of 2.

Programming Assignment `ternary_search`

- Write the function `unsigned int ternary_search( const std::vector< int > &data, int value )`.
- Add to the file `search.cpp` for this task.
- Verify that the function will correctly find the index of `value` within `data`.
- You may assume that `value` is present, and `data` is already sorted in ascending order.
- At the top of your source file, include a comment with your estimated Big-Oh complexity of the algorithm.
- `ternary_search` divides its input array into 3 equally sized groups, in the same way that `binary_search` divides into 2 equally sized groups.
- In the first pass of your code, write it to handle vectors whose sizes are powers of 3.
- In the second pass of your code, write it to handle vectors whose sizes are not powers of 3.

Programming Assignment Data Collection

- Time `binary_search` and `ternary_search` on vectors of sizes 2^0, 2^1, ..., 2^30.
- Be sure to do correct statistical data collection.
- Submit a table of the data collected, and declaration of which appears to be faster.

Programming Assignment Chart Data

- Chart the normalized runtimes of `binary_search` and `ternary_search`.
- Add to the chart curves for N^1/2, N^1/3, LOG_2(N), LOG_3(N) and 1.
- Submit the chart, and a statement discussing which algorithm has better Big-Oh, and which algorithm is faster.
- Save the document as `search-chart.pdf`.

Submission

- Submit you solutions by the due date and time. For written problems, your work and answers as a PDF to Canvas. For code, submit the source code to the class git repository. For tables and graphs, submit a PDF to Canvas.