Link of Text Book (CLRS) - <https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf>

**Mid Term List of Topics with Resources:**

1. Algorithm Basic Definition, History etc. (Link - <https://en.wikipedia.org/wiki/Algorithm>) – not needed for the exam, only used as an introductory knowledge to the course
2. How to write a Pseudocode – (Page – 20 of CLRS)
3. Linear Search Algorithm (Pseudocode) ([https://docs.google.com/document/d/1gEG2m\_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing)](https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing)%20)
4. Dry Running or Simulating Linear Search Algorithm
5. How to convert pseudocode to code and vice versa ([https://docs.google.com/document/d/1gEG2m\_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing)](https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing)%20)
6. Code of Linear Search Algorithm (https://docs.google.com/document/d/1gEG2m\_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing)
7. Correctness Proof using Loop Invariant
8. Correctness Proof of Linear Search (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
9. Modifying a well-known algorithm to develop a new algorithm, for example developing FIND-MAX-DAY algorithm by modifying the Linear Search algorithm (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
10. Dry Running or Simulating FIND-MAX-DAY algorithm
11. Correctness Proof of FIND-MAX-DAY algorithm using Loop Invariant (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
12. Pseudocode to code conversion of FIND-MAX-DAY algorithm
13. Developing FIND-MIN-DAY algorithm
14. Dry Running or Simulating FIND-MIN -DAY algorithm
15. Correctness Proof of FIND-MIN-DAY algorithm using Loop Invariant (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
16. Pseudocode to code conversion of FIND-MIN-DAY algorithm
17. Insertion Sort Algorithm for sorting array in ascending order and the code for the Insertion Sort algorithm (page 18 of CLRS and <https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
18. Modify the Insertion Sort Algorithm to sort the array in descending order
19. Dry Running or Simulation of Insertion Sort Algorithm
20. Correctness Proof of Insertion Sort Algorithm (Pages 19 and 20 of CLRS)
21. Finding time complexity or running time of an algorithm using Random Access Machine (RAM) method or model – example Insertion Sort (Page 26 of CLRS)
22. Finding Best Case and Worst Case Running Time or Time Complexity of an algorithm – example Insertion Sort (Pages 26 and 27 of CLRS)
23. Finding Big O or Order of Growth of the worst case running time of different algorithms both from the RAM model equation and directly from the code (Pages 28 and 29 of CLRS and <https://web.mit.edu/16.070/www/lecture/big_o.pdf>)
24. Comparison of different Big O values (<https://web.mit.edu/16.070/www/lecture/big_o.pdf> and <https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
25. Best Case and Worst Case running time of Linear Search (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
26. Bubble Sort (Ascending Order) Algorithm Code (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
27. Correctness Proof of Bubble Sort algorithm
28. Dry Run or Simulation of Bubble Sort algorithm
29. Code to Pseudocode of Bubble Sort algorithm
30. Bubble Sort Algorithm for sorting array in descending order
31. Algorithmic Design Techniques – Incremental and Divide and Conquer
32. Algorithm (Code) for finding the n-th term of the Fibonacci Series using the Divide and Conquer Algorithmic Design Technique (<https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/>) with dry running or simulation
33. Best Case and Worst Case running time of Bubble Sort (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
34. Merge Sort Algorithm and its recurrence equation (Pages 31-36 of CLRS and <https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
35. Solving recurrence equation using Recursion Tree method – example Merge Sort (Page – 38 of CLRS and <https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
36. Master Method for solving recurrence equations (Page 94 of CLRS)
37. Repeated Substitution Method for solving Recurrence equations (<https://www.youtube.com/watch?v=Ob8SM0fz6p0>)
38. Quick Sort Algorithm (<https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/tutorial/>)
39. Selection Sort Algorithm (<https://www.geeksforgeeks.org/selection-sort/amp/>) and Best and Worst Case time complexity for Selection Sort (<https://randerson112358.medium.com/analyze-the-selection-sort-algorithm-f36c18e7e4ec>)
40. Counting Sort (<https://www.programiz.com/dsa/counting-sort)>
41. Fractional Knapsack and Greedy Algorithmic Design Technique (<https://www.geeksforgeeks.org/fractional-knapsack-problem/>)
42. Optimization problem -finding the best solution from all feasible solutions - Minimization and Maximization, Greedy cannot solve optimization problem at all times (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
43. Divide and Conquer – 3 steps explained using Merge Sort (Page 30 of CLRS)
44. Best Case and Worst Case time complexities of different searching and sorting algorithms (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
45. Time Complexity and Space Complexity (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing>)
46. Asymptotic Notation – Big O, Big Omega and Big Theta (Page 45 of CLRS)
47. Comparison of different Big O values or worst case complexity values (<https://docs.google.com/document/d/1gEG2m_PQfWuGr5UmxYLdlITKV8OcbNPaUemFJ1S7X7U/edit?usp=sharing> and <https://web.mit.edu/16.070/www/lecture/big_o.pdf>)