FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
On the practical task No. 8
“Practical analysis of advanced algorithms”

Performed by
Denis Zakharov,
Maxim Shitilov
Academic group J4132c
Accepted by
Dr Petr Chunaev

St. Petersburg
2022

Goal

Practical analysis of advanced algorithms.

Formulation of the problem

1. Choose an algorithm (interesting to you and not considered in the course) from the above-mentioned book sections.
2. Choose an algorithm interesting to you and proposed at most 10 years ago in a research paper for solving a certain practical problem (including optimization algorithms, graph algorithms, etc.).
3. Analyze the chosen algorithms in terms of time and space complexity, design technique used, etc. Implement the algorithms (or use the existing ones from the research paper) and produce several experiments. Your experiments should differ of those in the research paper. Analyze the results

Brief theoretical part

Let's briefly discuss problems, ideas and algorithms that we are going to use in this research:

Divide and Conquer Algorithm

This is another effective way of solving many problems. In Divide and Conquer algorithms, divide the algorithm into two parts; the first parts divide the problem on hand into smaller subproblems of the same type. Then, in the second part, these smaller problems are solved and then added together (combined) to produce the problem's final solution. Merge sorting, and quick sorting can be done with divide and conquer algorithms.

Naive, D&C Recursive and Strassen's algorithms for matrix multiplication from the Introduction to Algorithms, 4rd Edition Naive

Naive

If you have seen matrices before, then you probably know how to multiply them. If $A = (a_{ij})$ and $B = (b_{ij})$ are square $n \times n$ matrices, then in the product $C = A \times B$, we define the entry c_{ij}, for $i, j = 1, 2, ..., n$, by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Equation 1 — Value of each cell from matrix product

Naive Matrix Multiplication Time & Space Complexities are $O(n^2)$

However we can use here a Divide & Conquer approach plus the Strassen's method to slightly improve the multiplication process.

D&C

To keep things simple, when we use a divide-and-conquer algorithm to compute the matrix product $C = A \times B$,we assume that n is an exact power of 2 in each of the $n \times n$ matrices. We make this assumption because in each divide step, we will divide $n \times n$ matrices into four $n/2 \times n/2$ matrices, and by assuming that n is an exact power of 2, we are guaranteed that as long as $n \geq 2$, the dimension $n/2$ is an integer.

Suppose that we partition each of A, B, and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

Equation 2 — Matrix partitioning

so that we rewrite the equation $C = A \times B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Equation 3 — Matrix product

Equation above corresponds to the four equations

$$\begin{aligned} C_{11} &= A_{11} \times B_{11} + A_{12} \times B_{21} \\ C_{12} &= A_{11} \times B_{12} + A_{12} \times B_{22} \\ C_{21} &= A_{21} \times B_{11} + A_{22} \times B_{21} \\ C_{22} &= A_{21} \times B_{12} + A_{22} \times B_{22} \end{aligned}$$

Equation 4 — Parts of the product

Each of these four equations specifies two multiplications of $n/2 \times n/2$ matrices and the addition of their $n/2 \times n/2$ products. This D&C approach gives us the recurrence for the running time of SQUARE-MATRIX-MULTIPLY-RECURSIVE:

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Equation 5 — Running time of D&C approach

That's where the key to Strassen's method becoming a lifesaver is because it makes the recursion tree slightly less bushy. That is, instead of performing eight recursive multiplications of

$n/2 \times n/2$ matrices, it performs only seven. The cost of eliminating one matrix multiplication will be several new additions of $n/2 \times n/2$ matrices, but still only a constant number of additions. As before, the constant number of matrix additions will be subsumed by,Θ-notation (yes that's Big theta) when we set up the recurrence equation to characterize the running time.

Strassen's method is not at all obvious. (This might be the biggest understate- ment in this book.) It has four steps:

1. Divide the input matrices A and B and output matrix C in to $n/2 \times n/2$ submatrices, as in Equation 2. This step takes $\Theta(1)$ time by index calculation, just as in SQUARE-MATRIX-MULTIPLY-RECURSIVE.
2. Create 10 matrices $S_1, S_2, ..., S_{10}$,each of which is $n/2 \times n/2$ and the sum or difference of two matrices created in step 1. We can create all 10 matrices in $\Theta(n^2)$ time.
3. Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, ..., P_{10}$. Each matrix P_i is $n/2 \times n/2$.
4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. We can compute all four submatrices also in $\Theta(n^2)$ time.

We shall see the details of steps 2–4 in a moment, but we already have enough information to set up a recurrence for the running time of Strassen's method. Let us assume that once the matrix size n gets down to 1, we perform a simple scalar mul- tiplication, just as in line 4 of SQUARE-MATRIX-MULTIPLY-RECURSIVE. When n > 1, steps 1, 2, and 4 take a total of $\Theta(n^2)$ time, and step 3 requires us to per- form seven multiplications of $n/2 \times n/2$ matrices. Hence, we obtain the following recurrence for the running time $T(n)$ of Strassen's algorithm

$$T(n) = 7T(n/2) + \Theta(n^2)$$

Equation 6 — Running time of Strassen's algorithm

Thus, we see that Strassen's algorithm, comprising steps 1–4, produces the correct matrix product and that recurrence Equation 6 characterizes its running time. This recurrence has the solution $T(n) = \Theta(n^{lg\,7})$, Strassen's method is asymptotically faster than the straightforward SQUARE-MATRIX-MULTIPLY procedure.

Traveling Salesman Problem using the Self-Organizing Map

The Traveling Salesman Problem is a well known challenge in Computer Science: it consists on finding the shortest route possible that traverses all cities in a given map only once. Although its simple explanation, this problem is, indeed, NP-Complete. This implies that the difficulty to solve it increases rapidly with the number of cities, and we do not know in fact a general solution that solves the problem. For that reason, we currently consider that any method able to find a sub-optimal solution is generally good enough (we cannot verify if the solution returned is the optimal one most of the times). To solve it, we can try to apply a modification of the Self-Organizing Map (SOM) technique. Let us take a look at what this technique consists, and then apply it to the TSP once we understand it better.

Clustering from the Introduction to Algorithms, 4rd Edition 33.1

Suppose that you have a large number of data points (examples), and you wish to group them into classes based on how similar they are to each other. For example, each data point might represent a celestial star, giving its temperature, size, and spectral characteristics. Or, each data point might represent a fragment of recorded speech. Grouping these speech fragments appropriately might reveal the set of accents of the fragments. Once a grouping of the training data points is found, new data can be placed into an appropriate group, facilitating star-type recognition or speech recognition. These situations, along with many others, fall under the umbrella of clustering. The input to a clustering problem is a set of n examples (objects) and an integer k, with the goal of dividing the examples into at most k disjoint clusters such that the examples in each cluster are similar to each other. The clustering problem has several variations. For example, the integer k might not be given, but instead arises out of the clustering procedure. In this section we presume that k is given.

Lloyd's procedure

Lloyd's procedure just iterates two operations—assigning points to clusters based on the nearest-center rule, followed by recomputing the centers of clusters to be their centroids—until the results converge.

Input: A set S of points in $\mathbb{R}^d$, and a positive integer k.

Output: A k-clustering $\langle S^{(1)}, S^{(2)}, \ldots, S^{(k)}\rangle$ of S with a sequence of centers $\langle c^{(1)}, c^{(2)}, \ldots, c^{(k)}\rangle$.

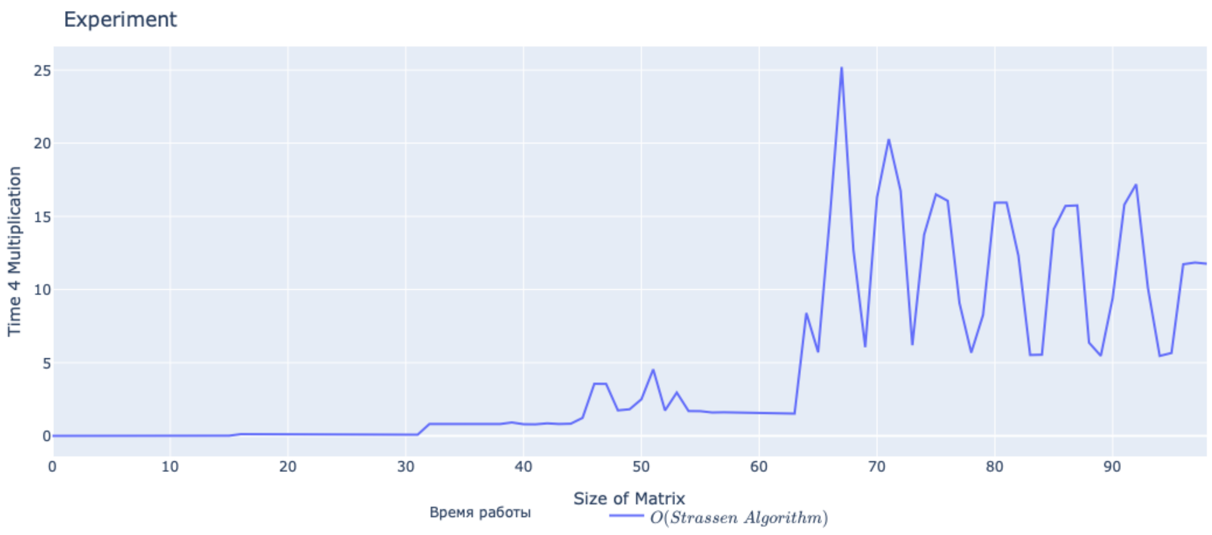
Here is Lloyd's procedure:

1. Initialize centers: Generate an initial sequence $\langle c^{(1)}, c^{(2)}, \ldots, c^{(k)}\rangle$ of k centers by picking k points independently from S at random. (If the points are not necessarily distinct, see Assign all points to cluster $S^{(1)}$ to begin.
2. Assign points to clusters: Use the nearest-center rule to define the clustering $\langle S^{(1)}, S^{(2)}, \ldots, S^{(k)}\rangle$. That is, assign each point $x \in S$ to a cluster $S(\ell)$ having a nearest center (breaking ties arbitrarily, but not changing the assignment for a point x unless the new cluster center is strictly closer to x than the old one).
3. Stop if no change: If step 2 did not change the assignments of any points to clusters, then stop and return the clustering $\langle S^{(1)}, S^{(2)}, \ldots, S^{(k)}\rangle$ and the associated centers $\langle c^{(1)}, c^{(2)}, \ldots, c^{(k)}\rangle$. Otherwise, go to step 4.
4. Recompute centers as centroids: For ℓ = 1, 2, …, k, compute the center $c(\ell)$ of cluster $S(\ell)$ as the centroid of the points in$S(\ell)$. (If $S(\ell)$ is empty, let $c(\ell)$ be the zero vector.) Then go to step 2.

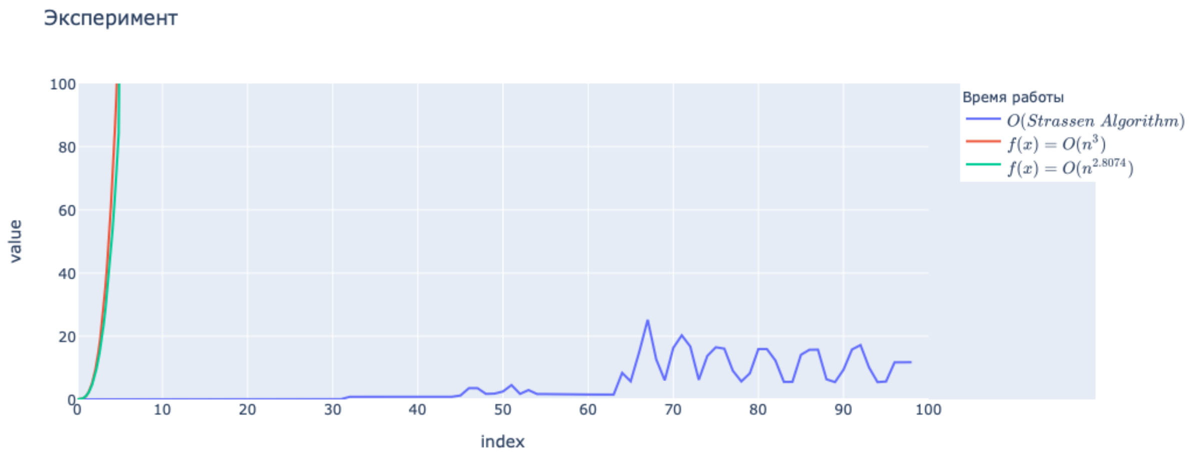
The running time of Lloyd's procedure is proportional to the number T of iterations. In one iteration, assigning points to clusters based on the nearest-center rule requires $O(dkn)$ time, and recomputing new centers for each cluster requires $O(dn)$ time (because each point is in one cluster). The overall running time of the k-means procedure is thus $O(Tdkn)$.

Results

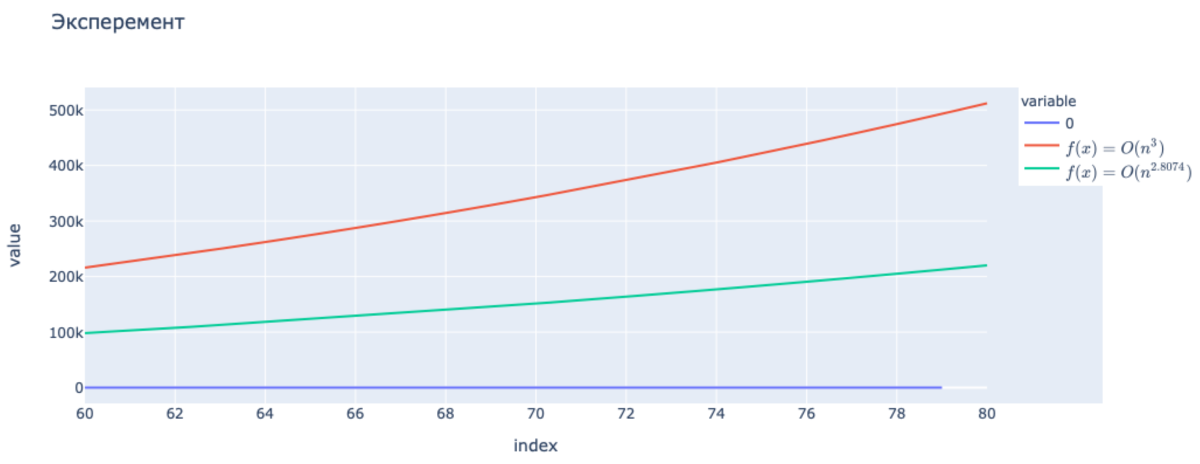
I Strassen's algorithms



Pic. 1 — Running time

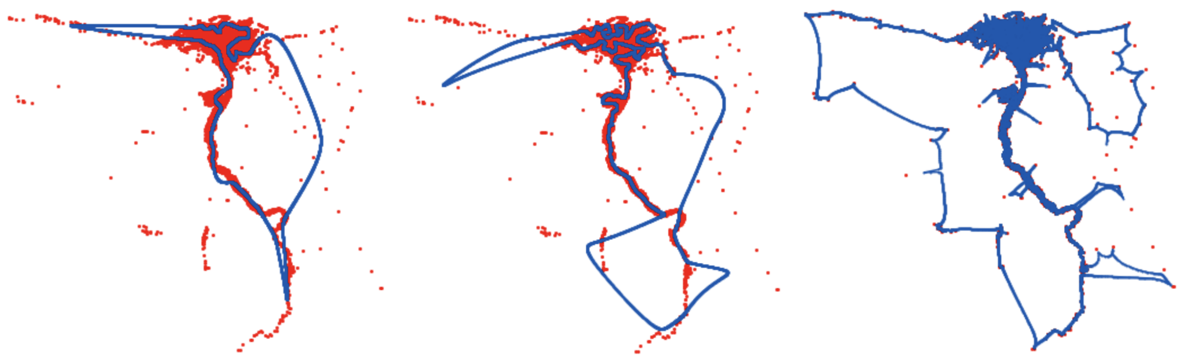


Pic. 2 — Running time

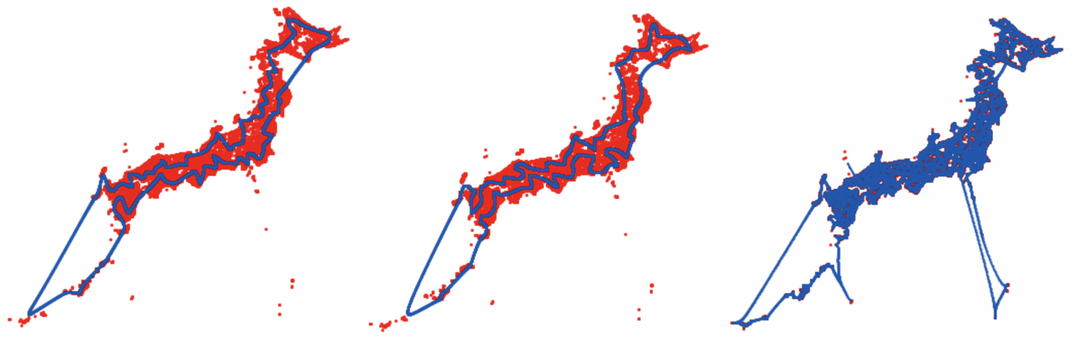


Pic. 3 — Running time

II Self-Organizing Maps for Traveling Salesman Problem



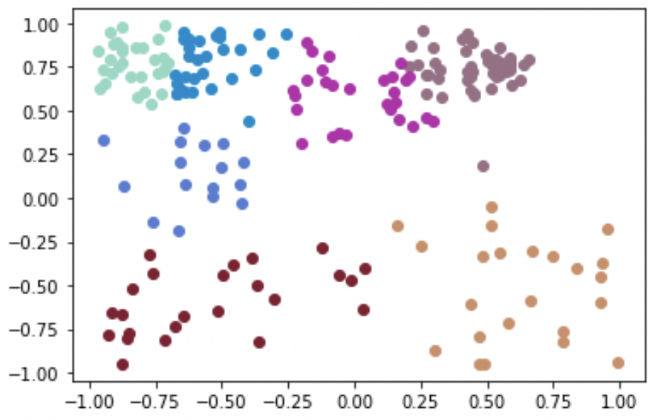
Pic. 4 — Egypt: 7146, 10_000, 100_000 iterations respectively



Pic. 5 — Japan: 9847, 10_000, 100_000 iterations respectively

III Clustering

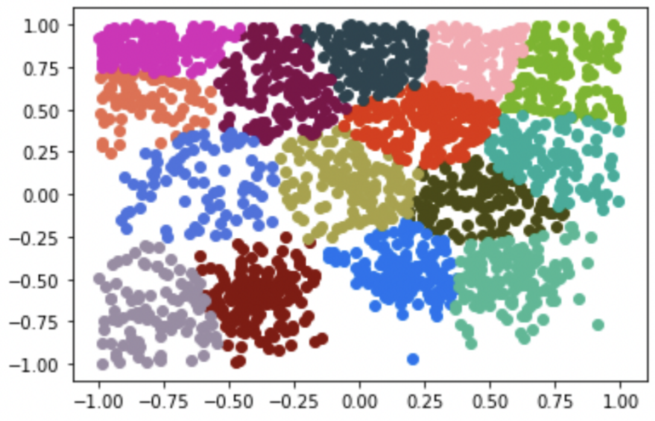
```
Time Before Start=2022-10-21 11:02:44.248761
Finished=2022-10-21 11:02:44.348796
0.100097 seconds
```



Pic. 6 — Lloyds procedure 200 points and 7 clusters

```
get_classification(2000, 15)
```

```
Time Before Start=2022-10-21 11:07:11.155539
Finished=2022-10-21 11:07:18.921656
7.766122 seconds
```



Pic. 7 — Lloyds procedure 2000 points and 15 clusters

Conclusions

In this research we got our hands onb practical analysis of advanced algorithms (Strassen, SOM for TSP, Lloyd’s k-Means for Clustering).

The Strassen algorithm is faster than the standard matrix multiplication algorithm for large matrices, with a better asymptotic complexity, although the naive algorithm is often better for smaller matrices, however it is also slower than the fastest known algorithms for extremely large matrices, but such galactic algorithms are not useful in practice, as they are much slower for matrices of practical size. For small matrices even faster algorithms exist.

To use the network to solve the TSP, the main concept to understand is how to modify the neighborhood function. If instead of a grid we declare a circular array of neurons, each node will only be conscious of the neurons in front of and behind it. That is, the inner similarity will work just in one dimension. Making this slight modification, the self-organizing map will behave as an elastic ring, getting closer to the cities but trying to minimize the perimeter of it thanks to the neighborhood function. It's worth to mention that SOM approach is enormously faster that the algorithm we were using in 5 laboratry.

Lloyd's k-Means is the most widely known Implementation of k-Means. When students first come into contact with clustering, they almost certainly learn Lloyd's k-Means. This is, as Lloyd is both, easy to understand and easy to implement. Let's get the Elephant out of the room, and look how this algorithm actually works. Lloyd initially chooses k datapoints as centroids, at random. After that, all datapoints are assigned to their closest centroid following the recalculation of centroids as mean over all their assigned datapoints. Lloyd loops over these last two steps until no point changes its assignment. Lloyd's algorithm illustrates an approach common to many machine-learning algorithms: First, define a hypothesis space in terms an appropriate sequence θ of parameters, so that each θ is associated with a specific hypothesis hθ . (For the k-means problem, θ is a dk-dimensional vector, equivalent to C, containing the d-dimensional center of each of the k clusters, and $h\theta$ is the hypothesis that each data point x should be grouped with a cluster having a center closest to x.) Second, define a measure $f(E,\ \theta)$ describing how poorly hypothesis hθ fits the given training data E. Smaller values of $f(E,\ \theta)$ are better, and a (locally) optimal solution (locally) minimizes $f(E,\ \theta)$. (For the k-means problem, $f(E,\ \theta)$ is just $f(S,\ C)$.) Third, given a set of training data E, use a suitable optimization procedure to find a value of θ * that minimizes$f(E,\ \theta\ *)$, at least locally. (For the k-means problem, this value of θ * is the sequence C of k center points returned by Lloyd's algorithm.) Return θ * as the answer.

Appendix

DataLore: site. – URL: https://datalore.jetbrains.com/notebook/RemqSkuJwmr1PM4Gc3cBqB/e9IoN4LKSiqIEuMqLboHYk (circulation date: 20.10.2022)

T. H. Cormen Strassen's algorithm for matrix multiplication / T. H. Cormen // Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. – 4th ed.. – Cambridge, Massachusetts London, England : MIT Press and McGraw-Hill., 2022. – 4.2. – C. 75-83. – ISBN 0-262-04630-X

T. H. Clustering / T. H. Cormen // Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. – 4th ed.. – Cambridge, Massachusetts London, England : MIT Press and McGraw-Hill., 2022. – 33.1. – C. 1296-1308. – ISBN 0-262-04630-X

World TSP: site. – URL: https://www.math.uwaterloo.ca/tsp/world/index.html (circulation date: 20.10.2022)

Using Self-Organizing Maps to solve the Traveling Salesman Problem // Diego Vicente : site. – URL: https://diego.codes/post/som-tsp/ (circulation date: 20.10.2022)

利用自组织映射解决旅行推销员问题 // WeChat : site. – URL: https://mp.weixin.qq.com/s/O7UHeTFfcJ1FjNShVe9wtA (circulation date: 20.10.2022)

Abdul Bari / 2 Divide And Conquer // YouTube : site. – URL: https://www.youtube.com/watch?v=2Rr2tW9zvRg (circulation date: 20.10.2022)

Abdul Bari / 2.9 Strassens Matrix Multiplication // YouTube : site. – URL: https://www.youtube.com/watch?v=0oJyNmEbS4w (circulation date: 20.10.2022)

Abdul Bari / 4.7 Traveling Salesperson Problem - Dynamic Programming // YouTube : site. – URL: https://www.youtube.com/watch?v=XaXsJJh-Q5Y (circulation date: 20.10.2022)

Abdul Bari / 4.7 [New] Traveling Salesman Problem - Dynamic Programming using Formula // YouTube : site. – URL: https://www.youtube.com/watch?v=Q4zHb-Swzro (circulation date: 20.10.2022)

Abdul Bari / 7.3 Traveling Salesman Problem - Branch and Bound // YouTube : site. – URL: https://www.youtube.com/watch?v=1FEP_sNb62k (circulation date: 20.10.2022)