**Divide and Conquer**

**Divide and conquer** гэдэг нь том асуудлыг жижиг, энгийн дэд хэсэг болгон задлах, дэд бодлогыг рекурсив аргаар шийдвэрлэх, шийдлүүдийг нэгтгэн анхны асуудлыг шийдэх арга юм.

Энэ арга нь үр ашигтай, энгийн, модульчлогдсан, параллель байдал, кэшэд ээлтэй байдлаараа хэд хэдэн давуу талтай ба нь үүнийг компьютерын шинжлэх ухаан, математикийн өргөн хүрээний асуудлыг шийдвэрлэх үнэ цэнтэй хэрэгсэл болгодог. Энэхүү аргыг илүү сайн тайлбарлавал дараах жишээг авч үзэж болно.

Танд олон жижиг хэсгээс бүрдсэн зураг байгаа гэж төсөөлье. Энэ нь олон хэсэгтэй бөгөөд бүгдийг нэг дор хийхэд үнэхээр хэцүү байх нь мэдээж.

**Divide**: Эхлээд та зургаа олон хэсэг болгон хуваана.

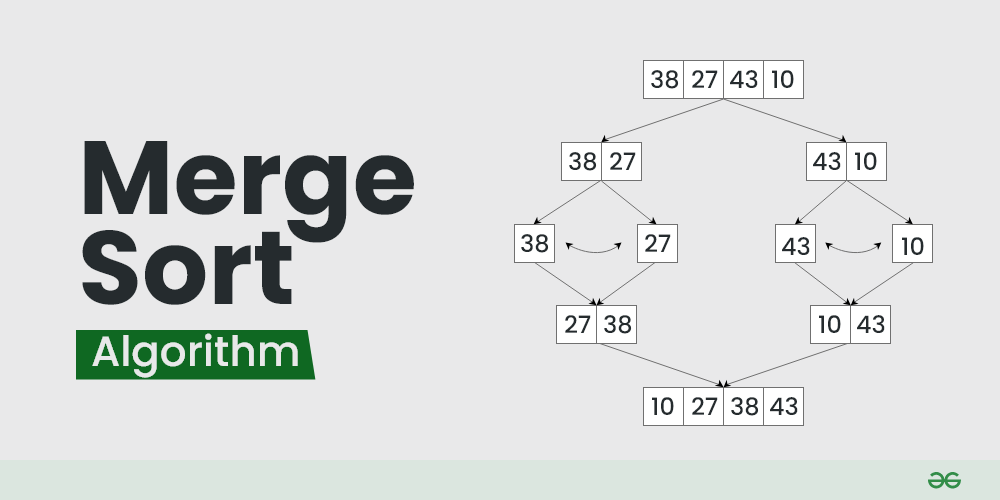
**Conquer**: Одоо та нэг нэг хэсэгт анхаарлаа хандуулаарай. Дараа нь хэсэг хэсгээр нь эвлүүлнэ.

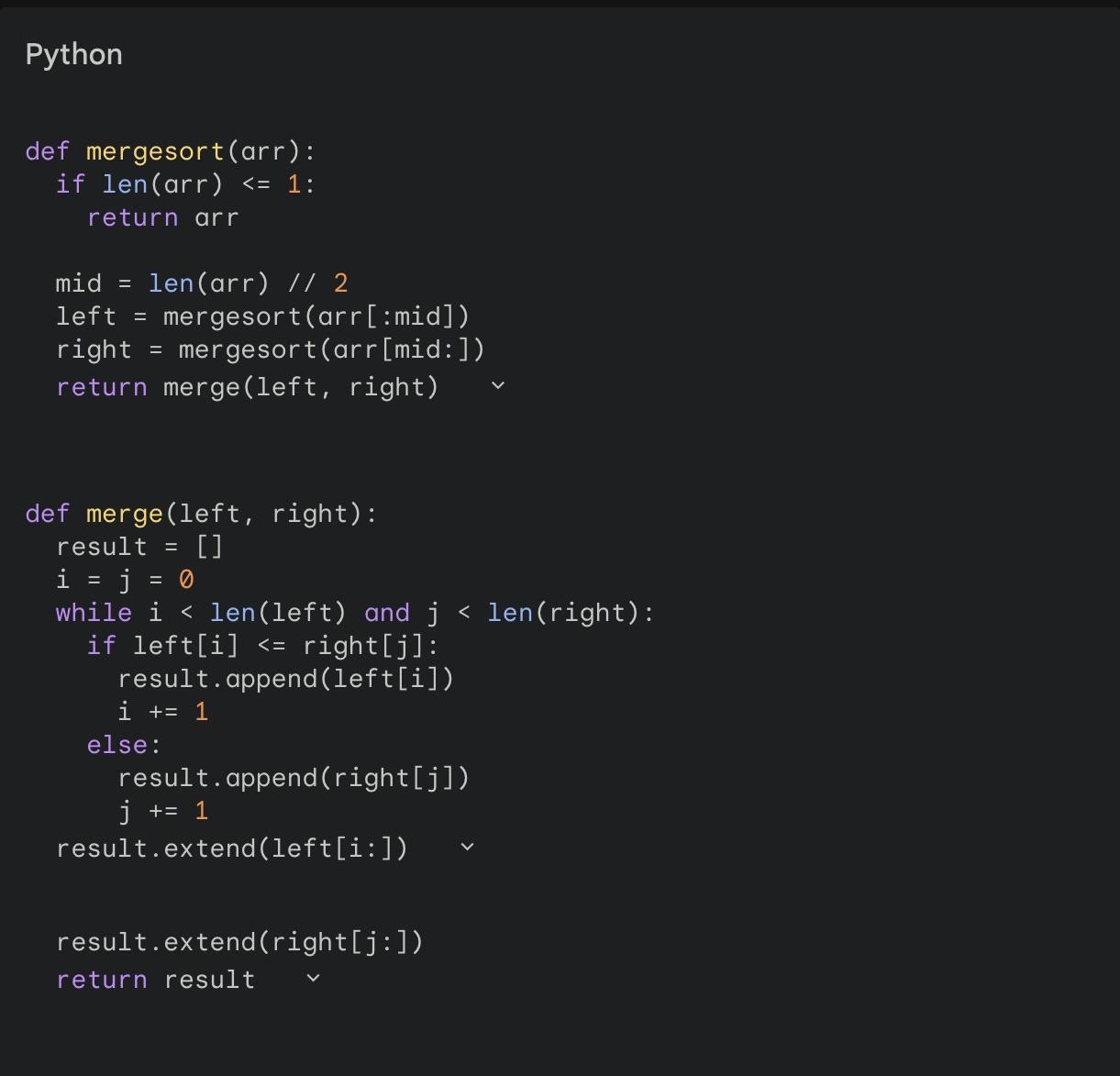
**Нэгтгэх:** Хэсэг бүрийг дуусгасны дараа та бүх хэсгүүдийг нийлүүлж улмаар зураг тань бүтэн болно.

Тус аргыг ихэвчлэн доорх тохиолдлуудад ашиглана.

* **Sorting algorithms (Эрэмбэлэлт):** Quicksort, mergesort, ба heapsort.
* **Searching algorithms (Хайлт):** Binary search (хамгийн түгээмэл ашиглагддаг)
* **Matrix multiplication (Mатриц):** Strassen's algorithm
* **Closest pair problem (Хамгийн ойр хос):** Энэ асуудал нь багц цэгүүдээс хамгийн ойр хоёр цэгийг олох явдал юм. Энэ асуудлыг үр дүнтэй шийдвэрлэхийн тулд хувааж, ялах аргыг ашиглаж болно.

Доор жишээ код оруулав.





**figure1.1** **figure1.2**

**Dynamic Programming**

**Dynamic programming** гэдэг нь том асуудлыг дэд асуудлууд болгон задлах, дэд асуудлуудыг шийдвэрлэж, шийдлүүдийг нэгтгэн анхны асуудлыг шийдэх арга юм. Энэ аргын гол онцлог нь дэд асуудлуудын шийдлүүдийг дахин ашиглах явдал юм. Dynamic programming нь ихэнхдээ дахин тооцохоос зайлсхийж, хугацаа, орон зайны үр ашгийг сайжруулахад тусалдаг. Энэ аргыг илүү сайн тайлбарлавал дараах жишээг авч үзэж болно.

**Problem**: Танд олон жижиг хэсгээс бүрдсэн зураг байгаа гэж төсөөлье. Энэ нь олон хэсэгтэй бөгөөд бүгдийг нэг дор хийхэд үнэхээр хэцүү байх нь мэдээж.

**Divide**: Эхлээд та зургаа олон хэсэг болгон хуваана.

**Solve small piece**: Одоо та нэг нэг хэсэгт анхаарлаа хандуулаарай. Дараа нь хэсэг хэсгээр нь эвлүүлнэ.

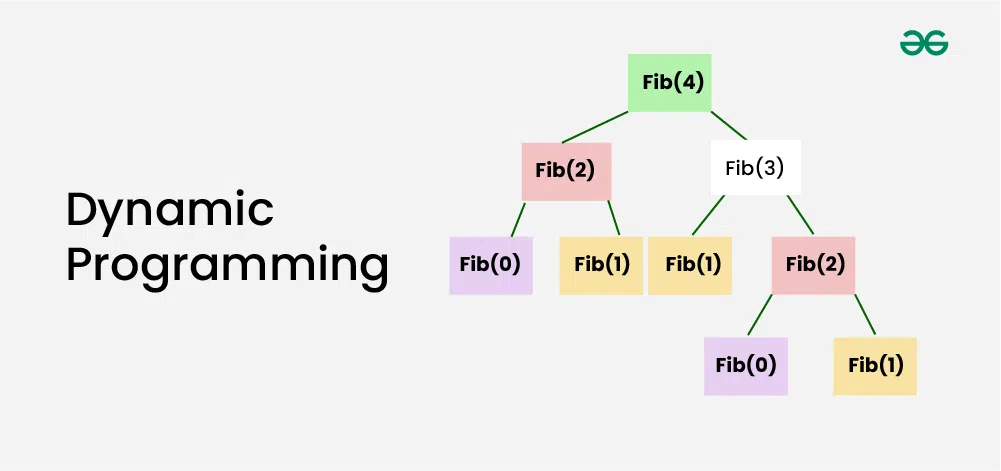
**Remember solution:** Өмнөх жижиг хэсгүүдээ эвлүүлэх явцдаа та туршлага хуримтлуулж улмаар дараагийн хэсгээ эвлүүлэхдээ илүү хурдан эвлүүлэх болно.

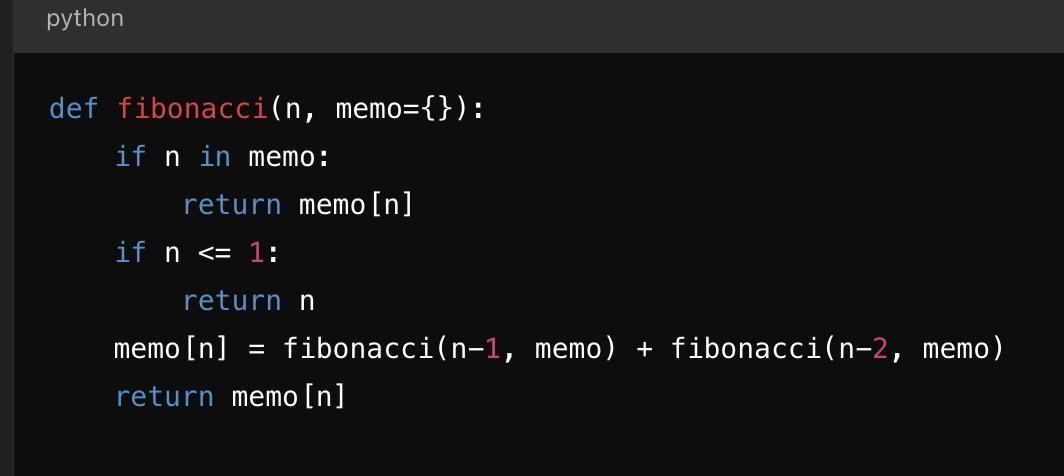
**Нэгтгэх:** Хэсэг бүрийг дуусгасны дараа та бүх хэсгүүдийг нийлүүлж улмаар зураг тань бүтэн болно.

Тус аргыг ихэвчлэн доорх тохиолдлуудад ашиглана:

* **Fibonacci sequence (Фибоначчийн тоон дараалал)**: Фибоначчийн тоог dynamic programming-аар тооцоолох.
* **Longest common subsequence (Хамгийн урт нийтлэг дэд дараалал):** Дэд дарааллуудын нийтлэг хэсгийг олох.
* **Knapsack problem (Багийн асуудал)**: Хэрэгцээтэй зүйлсийг багтаах хамгийн үр дүнтэй шийдлийг олох.
* **Matrix chain multiplication (Матрицын олонлог)**: Матрицын нийлбэрийг хамгийн бага зардлаар хэрхэн хийх вэ гэдгийг тодорхойлох.

Доор жишээ код оруулав.





**figure2.1**  **figure2.2**

**Greedy Programming**

**Greedy programming** нь алхам бүрдэх хамгийн сайн шийдлийг олох замаар үндсэн асуудлаа шийддэг арга юм. Гэвч энэ арга нь үндсэн асуудлаа харгалзалгүйгээр одоогийн алхам дахь хамгийн сайн шийдлийг олдгоороо онцлог. Энэ арга нь ихэвчлэн оновчлолын асуудлууд мөн алхам бүр дээрх хамгийн сайн шийдлийг олох зэрэг асуудлууд дээр ашиглагддаг. Гэхдээ нэг сул тал нь үргэлж хамгийн сайн шийдлийг олж өгдөггүйгээрээ онцлог. Энэхүү аргыг илүү сайн тайлбарлавал дараах жишээг авч үзэж болно.

Та тоглоом худалдаж авахын тулд зоос цуглуулах хэрэгтэй бөгөөд үүнийг аль болох хурдан хийхийг хүсэж байна гэж төсөөлье.

**Bigger coin goal**: Тоглоом худалдаж авахад 10 ширхэг зоос хэрэгтэй.

**Bigger coin 1st:** Танд байгаа зооснуудаасаа хамгийн ихийг нь авна.

**Next big goal:** Үүний дараа мөн хамгийн том зоосуудыг авна.

**Continue step**: Та 10 зоос хүртлээ хамгийн том зоосыг авсаар байна.

Тус аргыг ихэвчлэн доорх тохиолдлуудад ашиглана.

### **1.Coin Change Problem (Minimal Coins)**

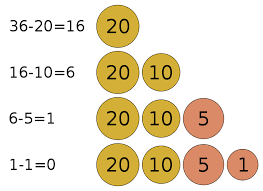
### **2. Activity Selection Problem (Maximize Non-Overlapping Activities)**

### **3. Fractional Knapsack Problem**

### **4. Huffman Coding (Data Compression)**

### **5. Prim’s Algorithm (Minimum Spanning Tree)**

### **6. Dijkstra’s Algorithm (Shortest Path in Graphs)**

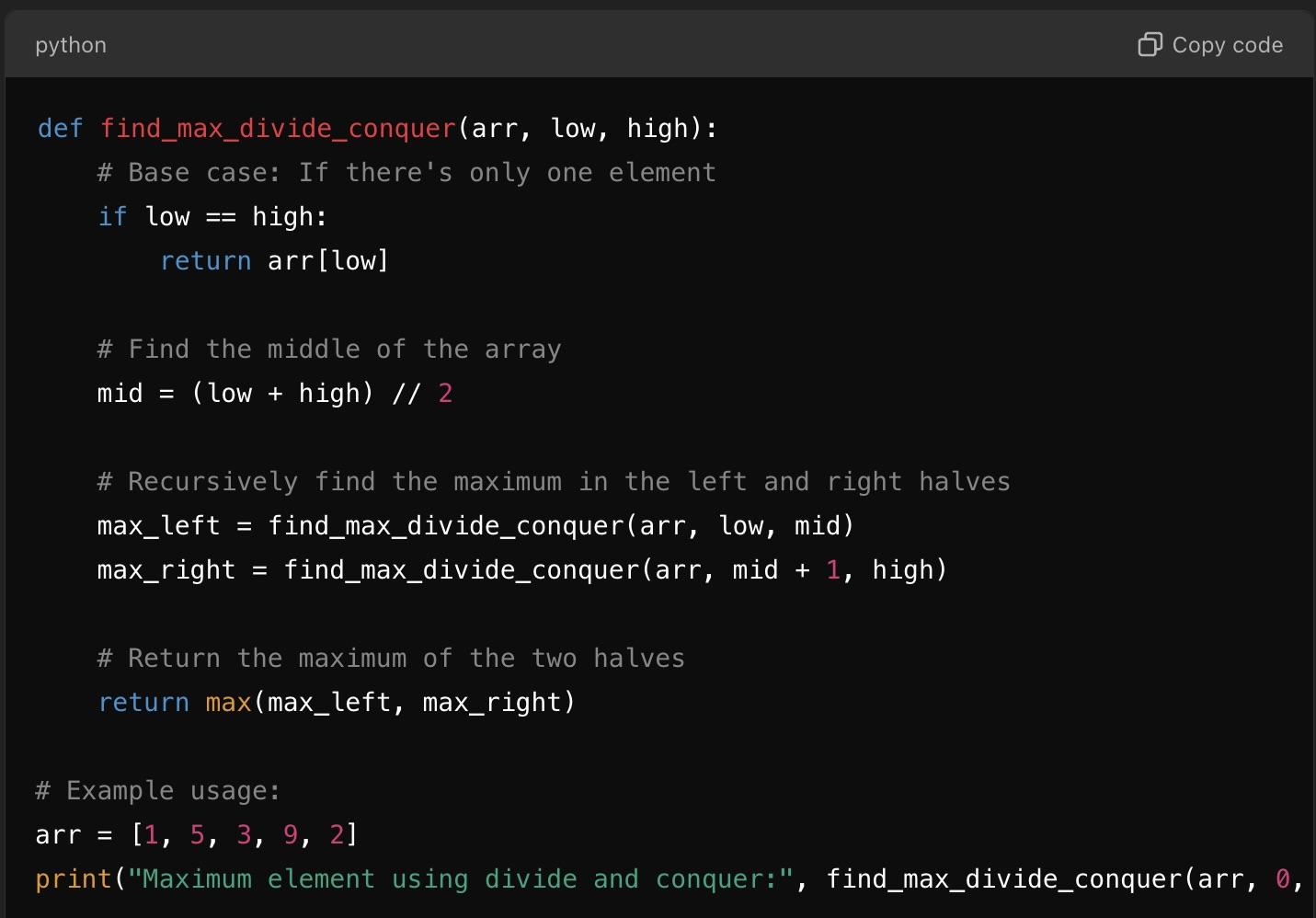
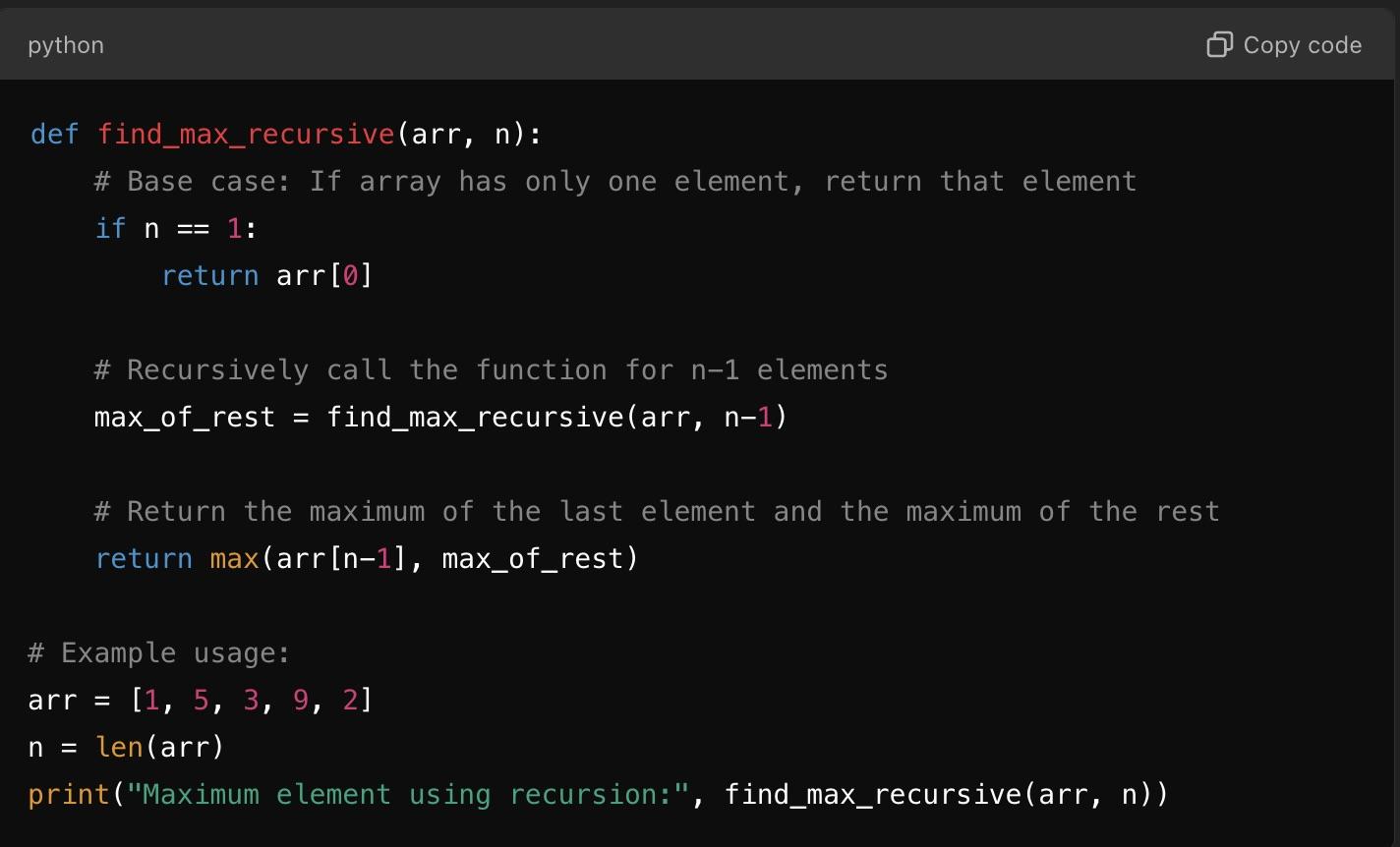


sdcsdcsdcsd


**figure3.1 figure3.2**

**Recursion vs Divide and Conquer**

| **Онцлог** | **Recursion** | **Divide and conquer** |
| --- | --- | --- |
| **давуу тал** | Энгийн бөгөөд ижил дэд асуудлуудтай бодлого дээр сайн ажиллана. | Томоохон асуудлуудыг жижиг хэсгүүд болон хувааж, төвөгтэй байдлыг багасгана. |
| **Сул тал** | Хэт их дуудалтын улмаас стек халилт зэрэг асуудлууд гарна. | Хэрэгжүүлэхэд төвөгтэй ба хуваан нэгтгэхэд зардал их гарна. |
| **хэрэглээ** | * Fibonacci sequence * Factorial * Tree traversals (preorder, postorder, inorder) * Depth-first search (DFS) in graphs | * Merge Sort * Quicksort * Binary Search * Matrix Multiplication (Strassen’s Algorithm) * Fast Fourier Transform (FFT) |
| **сайжруулалт** | Memoization ашиглан сайжруулж болно. | Dynamic programming ашиглан сайжруулж болно. |
| **дэд асуудал** | Дэд асуудал нь ижил | Бие даасан дэд асуудал |

****

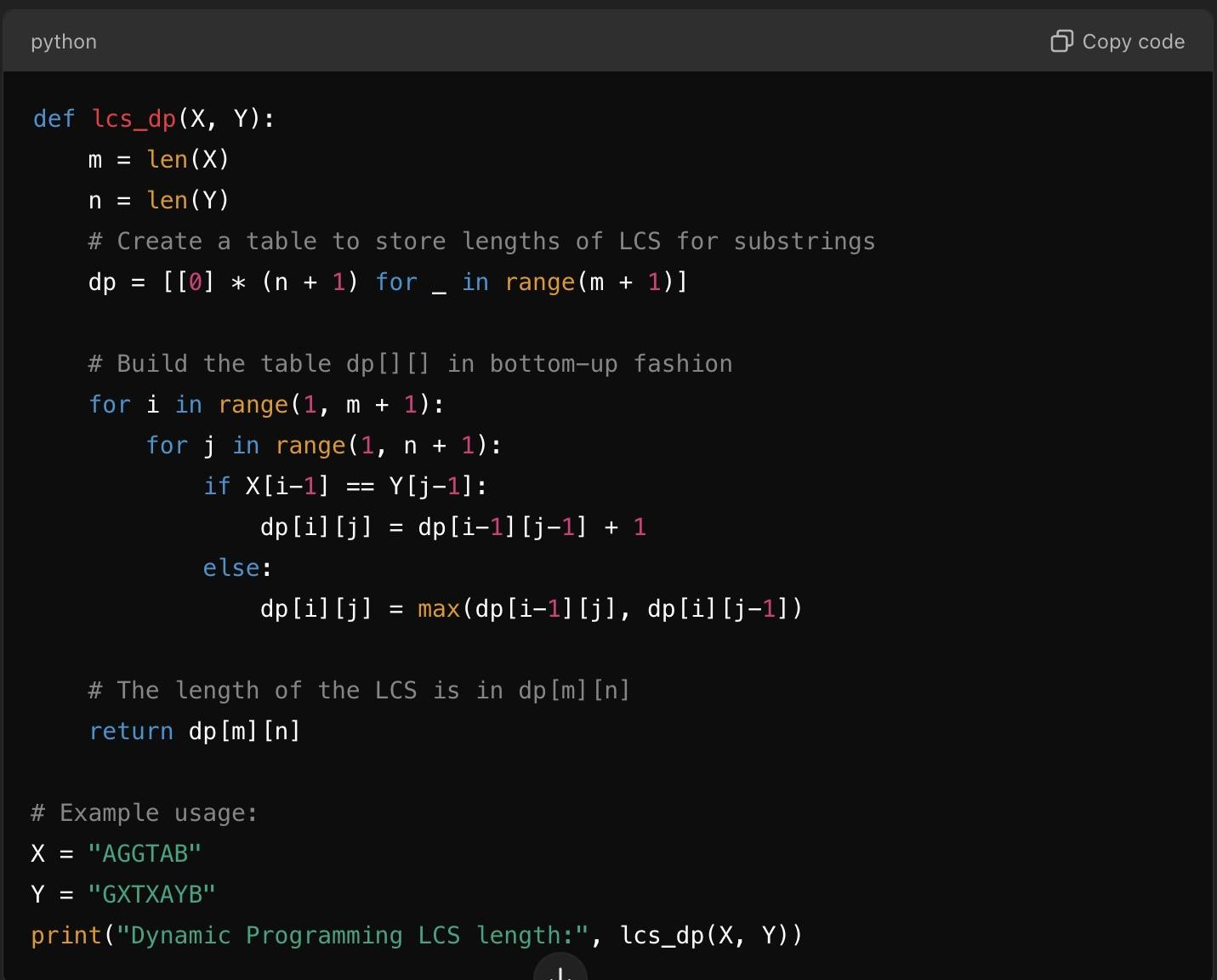
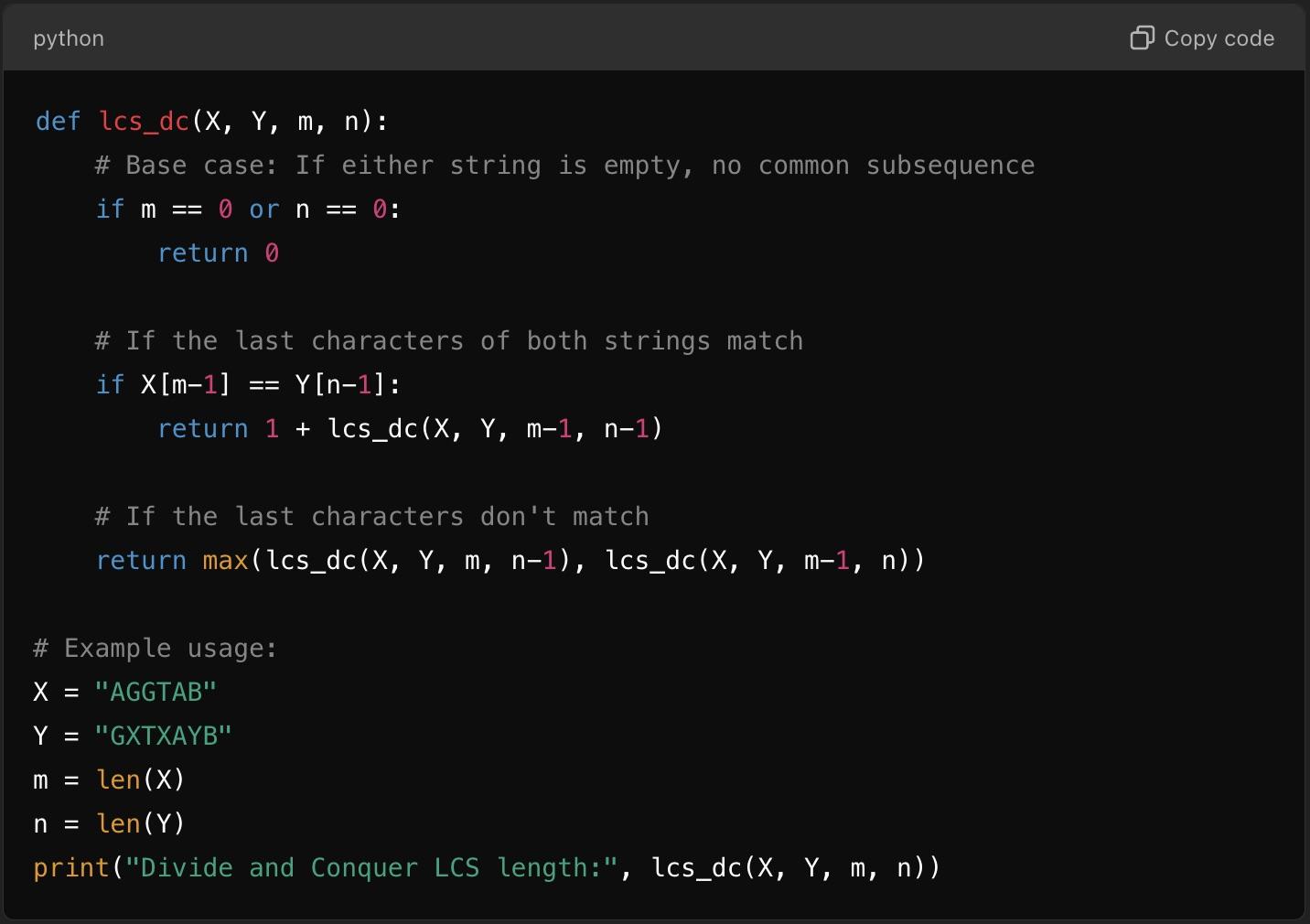
**figure4.1 figure4.2**

Дээрх жишээнд элементүүдээс хамгийн их тоог олдог бодлогыг recursive болон divide and conquer арга ашиглан хэрхэн бодож байгааг харуулав.

**Divide and conquer vs Dynamic programming**

|  | **Divide and conquer** | **Dynamic programming** |
| --- | --- | --- |
| **давуу тал** | Энгийн бөгөөд ижил дэд асуудлуудтай бодлого дээр сайн ажиллана. | Илүүдэл тооцоолол хийх зардлыг багасгана. |
| **Сул тал** | Хэт их дуудалтын улмаас стек халилт зэрэг асуудлууд гарна. | Санах ой их ашиглана |
| **Ажиллах хурд** | O(n log n) | O(n^2) or O(n^3) |
| **Санах ой ашиглалт** | O(log n) | O(n) or O(n^2) |
| **сайжруулалт** | Memoization ашиглан сайжруулж болно. | Tabulation |
| **Үндсэн ойлголт** | recursion | Memoization / Tabulation |

Доорх жишээнд 2 өөр текстед байгаа хамгийн урт текстийг гаргах бодлогыг дээрх 2 аргаар хэрхэн бодож байгааг харуулав.

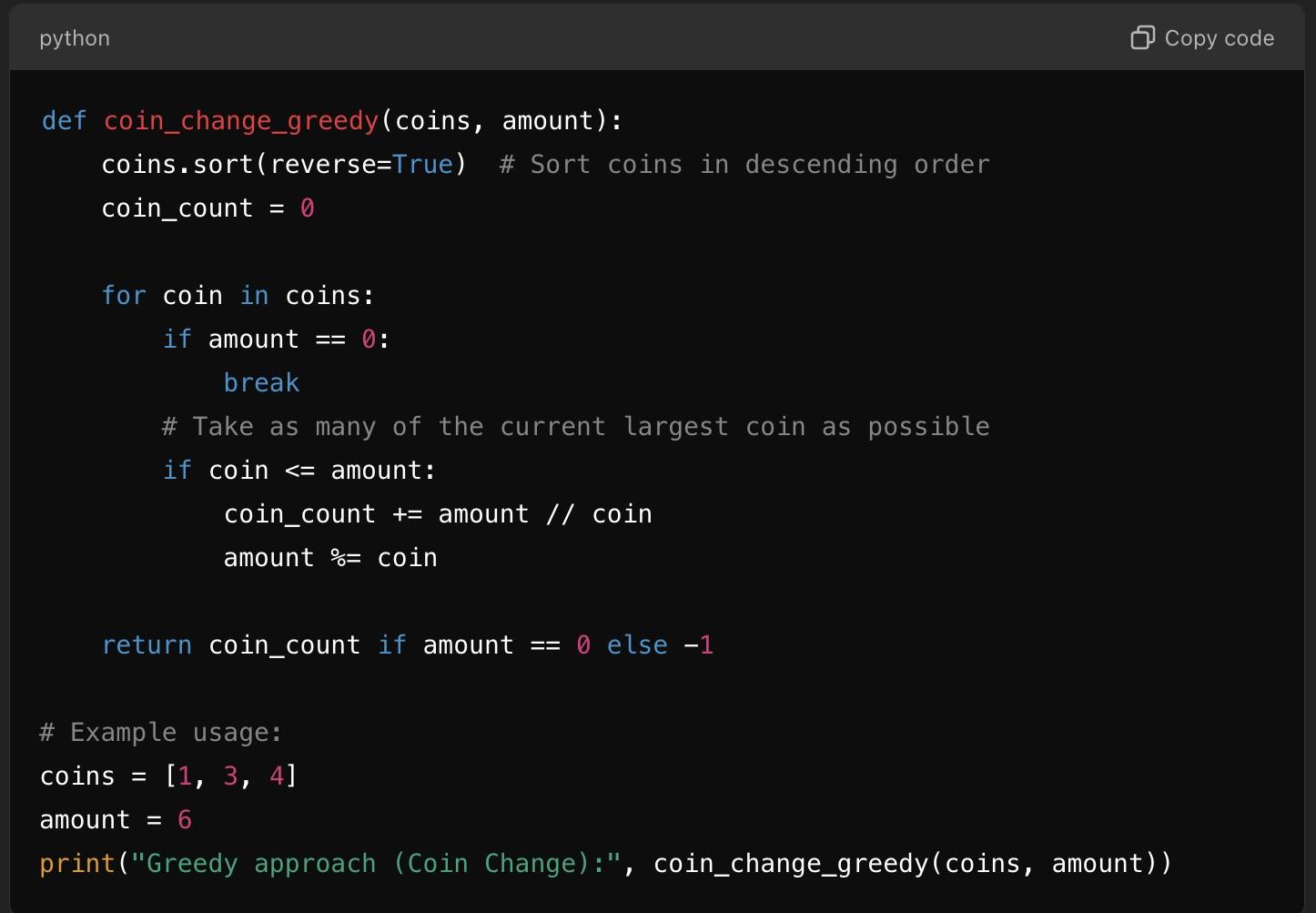
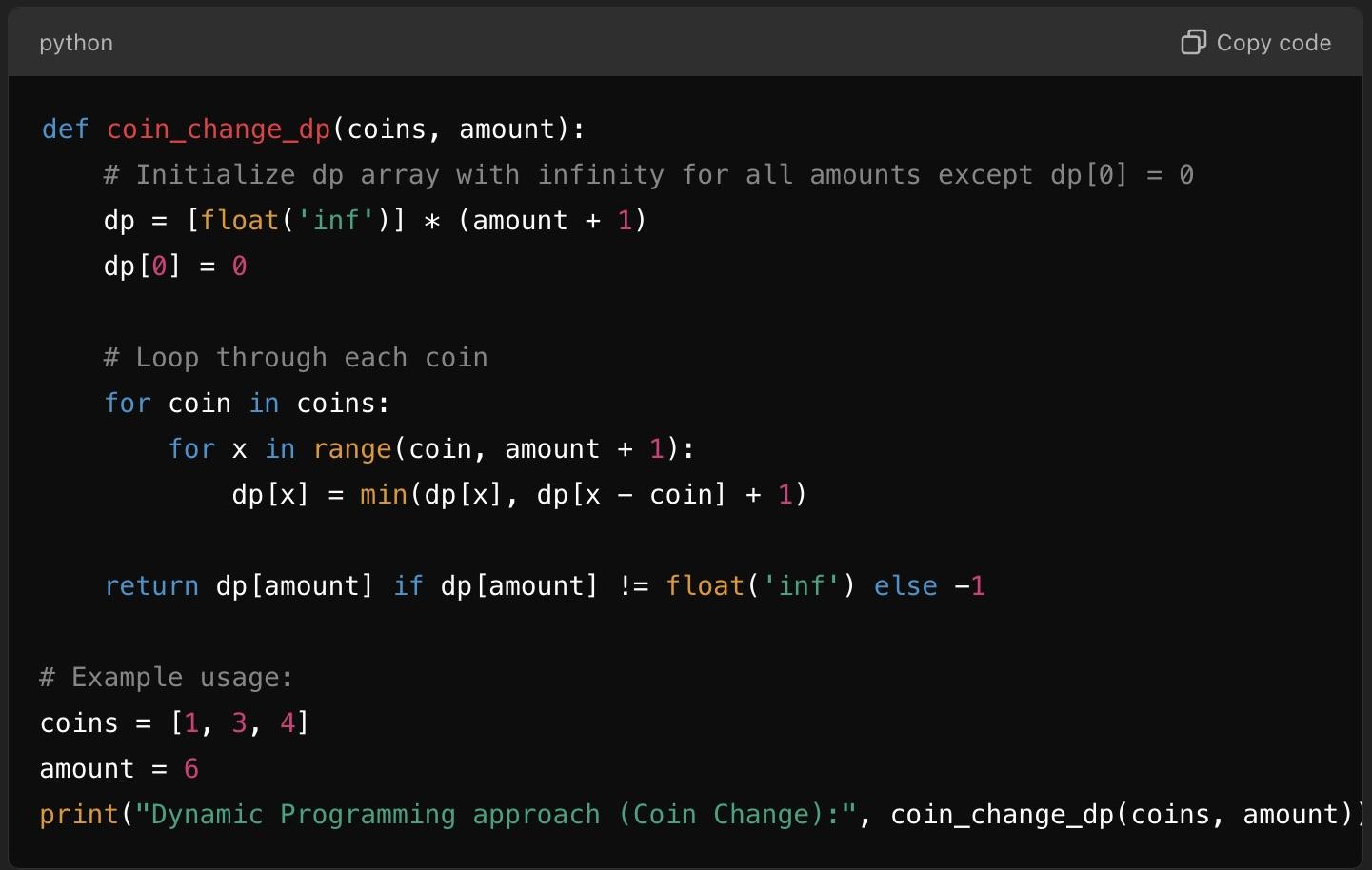
****

**figure5.1 figure5.2**

**Dynamic programming vs Greedy**

|  | **dynamic programming** | **greedy** |
| --- | --- | --- |
| **давуу тал** | Илүүдэл тооцоолол хийх зардлыг багасгана. | dynamic programming-аас илүү энгийн бөгөөд илүү сайн үр дүнд хүрч болно. |
| **Сул тал** | Санах ой их ашиглана | Үргэлж сайн шийдлийг өгнө гэсэн баталгаа байхгүй. |
| **Ажиллах хурд** | O(n^2) or O(n^3) | O(n log n) or O(n) |
| **Санах ой ашиглалт** | O(n) or O(n^2) | O(1) to O(n) |
| **сайжруулалт** | Tabulation | Хамгийн сайн сонголт |
| **Үндсэн ойлголт** | Memoization / Tabulation | Local Optimality |

Доорх жишээнд хамгийн бага зоос ашиглан үндсэн зоосонд хүрэх бодлогыг дээрх 2 аргаар хэрхэн бодож байгааг харуулав.



**figure6.1 figure6.2**

**Ашигласан материалууд**

1.Онолын хэсэг болон бусад жишээ хэсгүүдийг [chatgpt](https://chat.openai.com/chat) болон [gemini](https://gemini.google.com/) ашиглан бэлдэв.

1.

* figure1.1 эх сурвалж: https://chat.openai.com/chat
* figure1.2 эх сурвалж: https://media.geeksforgeeks.org/wp-content/uploads/20230706153706/Merge-Sort-Algorithm-(1).png

2.

* figure2.1 эх сурвалж: https://chat.openai.com/chat
* figure1.2 эх сурвалж: https://media.geeksforgeeks.org/wp-content/uploads/20240319104901/dynamic-programming.webp

3.

* figure3.1 эх сурвалж: https://chat.openai.com/chat
* figure3.2 эх сурвалж: https://en.wikipedia.org/wiki/Greedy\_algorithm#/media/File:Greedy\_algorithm\_36\_cents.svg

4.

* figure4.1 эх сурвалж: https://chat.openai.com/chat
* figure4.2 эх сурвалж: https://chat.openai.com/chat

5.

* figure5.1 эх сурвалж: https://chat.openai.com/chat
* figure5.2 эх сурвалж: https://chat.openai.com/chat

6.

* figure6.1 эх сурвалж: https://chat.openai.com/chat
* figure6.2 эх сурвалж: https://chat.openai.com/chat