Blatt 10

Aufgabe 1

- Sortieren Sie das Array: $[5, 8, 3, 7, 1, 4, 2, 9, 6]$ manuell mit Quicksort.

  Pivot: $5 - 1 - 6 \Rightarrow 1 - 5 - 6 \Rightarrow \boxed{5}$

  1. $[1, 8, 3, 7, \boxed{5}, 4, 2, 9, 6]$ ← Pivot gewählt
  2. $[1, 3, 4, 2, \boxed{5}, 8, 7, 9, 6]$ ← kleinere Zahlen nach links und größere nach rechts
  3. $[1, 3, 4, \boxed{2},]\boxed{5}[, 8, 7, 9, 6]$ ← linke partition mit median sortieren
  4. $[1, \boxed{2}, 4, 3,]\boxed{5}[, 8, 7, 9, 6]$ ← pivot = 2
  5. $[1, 2, 4, \boxed{3},]\boxed{5}[, 8, 7, 9, 6]$ ← pivot = 3, swap 3 und 4
  6. $[1, 2, \boxed{3}, 4,]\boxed{5}[, 8, 7, 9, 6]$ ← lower half sortiert, nun die higher half
  7. $[1, 2, 3, 4, \boxed{5}[, 8, 7, 9, 6]$ ← pivot wählen aus median
  8. $[1, 2, 3, 4, \boxed{5}[, 6, \boxed{7}, 9, 8]$ ← pivot = 7
  9. $[1, 2, 3, 4, \boxed{5}[, 6, 7, 9, \boxed{8}]$ ← pivot = 8
  10. $[1, 2, 3, 4, \boxed{5}[, 6, 7, \boxed{8}, 9]$ ← swap 8 und 9
  11. $[1, 2, 3, 4, 5, 6, 7, 8, 9]$ ← sortierte Liste

- Beweise Sie per Gegenbeispiel, dass Quicksort nicht stabil ist.

  Array: $[5(1), 5(2), 5(3), 1, 4]$ → Pivot: $\boxed{4}$

  1. $[5(1), 5(2), 5(3), 1, \boxed{4}]$
  2. $[1, 5(2), 5(3), 5(1), \boxed{4}]$
  3. $[1, \boxed{4}, 5(3), 5(1), 5(2)]$

  Hier sieht man das nach dem Quicksort die Reihenfolge von den drei 5 nicht mehr stimmt, weshalb Quicksort auch instabil ist.

Aufgabe 2: Parallelisierung von Sortieralgorithmen

▼ parallelisierbar:

▼ Quicksort

▼ Kann parallel ausgeführt werden, da es den Divide and Conquer Paradigma nutzt. Dadurch kann die zu Sortierende Liste partitioniert werden und dann anschließlich unabhängig voneinander sortiert werden und zum Schluss wieder zusammengefügt werden. Beim Quicksortist jedoch die Effizienz stark von dem Pivotelement abhängig.

▼ Mergesort

▼ Auch Mergesort baut auf Divide and Conquer auf, weshalb Mergesort parallelisierbar ist. Die Liste wird bis in die einzelteile aufgeteilt und anschließend kann die List parallel zusammengeführt werden.

▼ nicht parallelisierbar:

▼ Selection Sort

▼ Selection Sort kann nicht parallelisiert werden, da die Liste immer komplett abgelaufen werden muss. Daher ist es nicht möglich die Liste aufzuteilen und dann wieder zusammenzuführen.

▼ Insertion Sort

▼ Auch Insertion Sort kann nicht parallelisiert werden, da die Liste sequenziell traversiert werden muss. Deshalb kann die List nicht durch aufteilen soretiert werden.

▼ Heap Sort

▼ Heap Sort kann nicht parallelisiert werden, da die Elemente einer List hintereinander eingefügt werden. Man kann nicht den Tree aufteilen und am Schluss zusammenführen da die Ordnung nicht mehr stimmen kann. Würde man den Baum aufteilen dann würden zwei unabhängige sortierte Bäume entstehen die nicht zusammengeführt werden können.

Aufgabe 3: Dynamische Programmierung

1. Beschreiben Sie die Funktion $g[m, n]$ zum besseren Verständnis kurz in natürlicher Sprache.
   a. Die Funktion $g[m, n]$ sagt aus, wenn keine Leistungspakete existieren oder keine Monate verfügbar sind, dann kann es keinen Gewinn geben.
   b. Wenn die Dauer der Leistungspakete mehr Monate in Anspruch nimmt als verfügbar, dann werden diese Pakete entfernt und die gewinnfunktion wird mit einem paket weniger aufgerufen.

c. Andernfalls wird der maximale Gewinn berechnet, indem man das Maximum der Monate ohne den Paketen, welche länger brauchen, und von den Gewinn, wo die Dauer der bearbeitung abgezogen wird pro Leistungsoaket plus den Einnahmen der Leistungspakete. So kann jede Kombination durchgerechnet werden und anschließend kann der maximale Gewinn ausgegeben werden.

2. $E = [4500, 6500, 13000]\ und\ M = [2, 3, 5], m = 12, n = 3$

m/n	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	4500	4500	4500
3	0	4500	6500	6500
4	0	9000	9000	9000
5	0	9000	11000	13000
6	0	13500	13500	13500
7	0	13500	15500	17500
8	0	18000	18000	19500
9	0	18000	20000	22000
10	0	22500	22500	26000
11	0	22500	24500	26500
12	0	27000	27000	30500

Für den maximalen Gewinn: $2 * 5Monate + 1 * 2Monate$

3. Man könnte einen Greedy Algorithm verwenden um am Schluss von den ganzen Geldbeträgen die beste Auswahl treffen um die 12 Monate zu füllen. Am besten vielleicht eine Map mit Key-Value als Monate und Value als Preise und den Greedy Algorithm auf die Keys anwenden.