Aufgabe 2.1 - Radixsort

$A_{(10)} = [231, 230, 101, 777, 450, 381, 950, 517]$
Alles in Darstellung zur Basis 6
$A_{(6)} = [1023, 1022, 0245, 3333, 2030, 1433, 4222, 2221]$

Phase	Verteilungsphase	Sammelphase(Array A)
1	Q 0: 2030 Q 1: 2221 Q 2: 1022, 4222 Q 3: 1023, 3333, 1433 Q 4: - Q 5: 0245	[2030, 2221, 1022, 4222, 1023, 3333, 1433, 0245]
2	Q 0: - ; Q 1: - Q 2: 2221, 1022, 4222, 1023 Q 3: 2030, 3333, 1433 Q 4: 0245 Q 5: -	[2221, 1022, 4222, 1023, 2030 ,3333, 1433, 0245]
3	Q 0: 1022, 1023, 2030 Q 1: - Q 2: 2221,4222, 0245 Q 3: 3333 Q 4: 1433 Q 5: -	[1022, 1023, 2030, 2221, 4222, 0245, 3333, 1433]
4	Q 0: 0245 Q 1: 1022, 1023, 1433 Q 2: 2030, 2221 Q 3: 3333 Q 4: 4222 Q 5: -	[0245, 1022, 1023, 1433, 2030, 2221, 3333, 4222]

Nach 4 Schritten (maximal 4 Stellen) ist das Array sortiert:
$A_{(6)} =$ [0245, 1022, 1023, 1433, 2030, 2221, 3333, 4222]
$A_{(10)} =$ [101, 230, 231, 381, 450, 517, 777, 950]

1.2 - Randomisierte Pivotwahl

Sei X_i die i-te gewählte Zahl ($i \in [1,7]$) und X_α die viertkleinste mit ($\alpha \in [1,7]$).
Gesucht ist P($n/4 < r(X_\alpha) \leq 3n/4$).
Untersuchen der Gegenereignisse:
Falls r(X_α) außerhalb liegt, tritt eines der folgenden 2 disjunkten Ereignisse ein.
Fall 1: Die mind. vier kleinsten Zahlen fallen alle an Positionen < n/4.
Fall 2: Die mind. vier größten Zahlen fallen alle an Positionen > 3n/4.
Da alle Elemente verschieden und uniform verteilt sind, gilt aus Symmetriegründen P(Fall 1) = P(Fall 2), denn mit Zurücklegen gilt:
$P(r(X_1) < n/4) = 1/4 = P(r(X_1) > 3n/4)$ (für alle X_i da unabhängig).
Fall 1 lässt sich durch die Binomialverteilung $B_{n,p}(3 < k \leq 7)$ mit n=7; p=1/4 und mind. 4 Treffer "in den kleineren Bereich darstellen".
P(Fall 1) =
$B_{7,1/4}(4) + B_{7,1/4}(5) + B_{7,1/4}(6) + B_{7,1/4}(7) \overset{T.R.}{=} 0,0705566$
Da Fall 1 und Fall 2 disjunkt sind folgt:
P(Fall 1 oder Fall 2) = 2 · 0,0705566 = 0,1411132.
Damit ergibt sich :
P($n/4 < r(X_\alpha) \leq 3n/4$) = 1- P(Fall 1 oder Fall 2) = 1-0,1411132 = 0.8588868.
b) Durch die gegebene Binomialverteilung lässt sich der Erwartungswert des ersten Misserfolgs als E[T] = $\frac{1}{1-p}$ berechnen. Dann ist E[T] = 1/(0,1411132) = 7,08.
$7 < E[T] < 8 \rightarrow$ Der erste Ausreisser wird nach 8 Pivotwahlen erwartet.

2.3 - Quickselect

Um die Problemgröße stets um 1 zu reduzieren, muss das pivotelement stets das größte oder kleinste aus dem restlichen Array sein.
Das lässt sich mit indexierung startend bei 0 so konstruieren: A= [a,b,c,d,e,f,g]
Schritt 1: Pivot= $A\lceil\frac{0+6}{2}\rceil$ = d, wähle d als maximum ergibt:
A = [**a,b,c,g,e,f** , d]
Schritt 2: Pivot= $A\lceil\frac{0+5}{2}\rceil$ = g, wähle g als rest-maximum ergibt:
A = [**a,b,c,f,e** , g,d]
Schritt 3: Pivot= $A\lceil\frac{0+4}{2}\rceil$ = c, wähle c als rest-maximum ergibt:
A = [**a,b,e,f** , c,g,d]
Schritt 4: Pivot= $A\lceil\frac{0+3}{2}\rceil$ = e, wähle e als minimum ergibt:
A = [e , **b,f,a**,c,g,d]
Schritt 5: Pivot= $A\lceil\frac{1+3}{2}\rceil$ = f, wähle f als rest-minimum ergibt:
A = [e,f , **a,b**,c,g,d]
Schritt 6: Pivot ist $A\lceil\frac{2+3}{2}\rceil$ = b , wähle b als rest-minimum ergibt:
A =[e,f,b, (a) , c,g,d] und der 7. Schritt findet a als viertgrößten Schlüssel.

Probe:
A = [4,b,c,7,1,f,g] = **[4,3,5,7,1,2,6]**
[4,3,5,7,1,2,6], pivot = 7 :
[(4,3,5,6,1,2) , 7], pivot = 6:
[(4,3,5,2,1) , 6,7], pivot = 5:
[(4,3,1,2) , 5,6,7], pivot = 1:
[1, (3,2,4) , 5,6,7], pivot = 2:
[1,2 , (4,3) , 5,6,7], pivot = 3:
[1,2,3 , (4) , 5,6,7], pivot = 4:✓

2.4 - Array-Veränderungen

Da Elemente nur vergrößert wurden, lässt sich leicht in $O(n)$ prüfen, welche Elemente nicht mehr sortiert sind. Sei A_1 das veränderte Array, dann verschiebt man alle maximal k Elemente, die die Sortierung stören, in ein neues Array B. Alle übrigen in A_2, so ergibt sich ein neues Array B mit maximal k Elementen und ein sortiertes Array A_2. Das Veränderte Array A_2 ist nun wieder sortiert. B lässt sich in $O(k \cdot log\ k)$ sortieren. Damit ergeben sich zwei sortierte Teilarrays, die sich in $O(n)$ mergen lassen. Die Gesamtlaufzeit ist dann $O(n+n+k \cdot log\ k) = O(n + k \cdot log\ k)$
Pseudocode:

```
//Das veraenderte Array A1 mit laenge n
for (i=0 ; < n-1 ; i++){
    vergleiche paarweise i und Nachfolger
        if A[i]> A[i+1] {B.append(A[i])} //falsch sortiert
        else {A2.append(A[i])} // richtig sortiert
        }
A2.append(A[n-1]) //letztes Element ist max(A2)

mergesort(B) // A2 ist schon sortiert
return merge(A2,B)
```

Korrektheit: Das verteilen erhält auf A_2 durch den Abgleich die Sortierung. Mergesort(B) ist laut Vorlesung korrekt. Damit sind A_2 und B sortiert. Merge liefert auch korrekte Ergebnisse, dann ist das Ergebnis sortiert.

2.5 - Mergesort

M ist Hauptspeicher; $B = M^{1/8}$; $n = M^{15}$
a)
Die Mergephasen sind laut Vorlesung gegeben als :
pro Iteration $\Theta(M/B) = \Theta(M^{7/8})$ Teilfolgen zusammenmischen.
Es gibt $\Theta(n/m) = M^{14}$ Teilfolgen.
Also ergeben sich $\Theta(log_{M^{7/8}}(M^{14})) = 16 = \Theta(1)$ Mergephasen.
Laut Vorlesung gilt Anzahl Phasen $= O(log_{M^{7/8}}(M^{14})$, doch aufgrund der exakten abhängigen Wahl in dieser Aufgabe ergibt sich auch:
Anzahl Phasen $= \Theta(log_{M^{7/8}}(M^{14}) = \Theta(1)$
b)
Auch hier lässt sich die Formel der Vorlesung anpassen. Quicksort sortiert im Hauptspeicher, dafür sind zu Beginn $\Theta(n/B) = \Theta(M^{119/8})$ Zugriffe nötig. Danach in jeder Mergephase erneut $\Theta(n/B)$ Zugriffe.
Da die Anzahl an Iterationen konstant ist, ergeben sich $\Theta(n/B) = \Theta(M^{119/8})$ Festplattenzugriffe.
In Abhängigkeit von der Größe der zu sortierenden Folge ist dies umgeformt $\Theta((n^{1/15})^{119/8}) = \Theta(n^{119/120}) = O(n^{0,99167})$.
Das ist interessant, denn wenn der Speicherzugriff der maßgebliche Zeitfaktor des Sortierens ist, sortieren wir hier in der Zeit $o(n)$. (!!)