# Implementierter Algorithmus

Implementiert wurde ein inplace MSD-Radix Sortieralgorithmus mit $2^k$ Buckets. Dabei wird dieser rekursiv auf den Buckets aufgerufen. Unterschreitet ein Bucket die vorgegebene Schranke $l$, so wird vom MSD-Radix zu einem Robin-Hood Sortieralgorithmus gewechselt. Dieser ist nicht in-place implementiert.

## Inplace MSD-Radix

Zu gegebenem $n$, $k$ und Interval $[a, b)$ wird zunächst für jeden Bucket $b_i, i \in \{1...2^k\}$ gezählt, wie viele Elemente diesem zugeteilt werden. Daraus werden die Startpositionen $s_i$ der Buckets iterativ erstellt, wobei wir die Endpositionen $t_i := s_i$ initialisieren. Anschließend wird über das Array iteriert und die Elemente $e_i$ entsprechend in die Buckets wie folgt einsortiert, wobei $b_j$ der Bucket von $e_i$ ist:

- Fall 1: $i \in [s_j, t_j)$

  $e_i$ ist bereits im richtigen Bucket, also wird das Ende des Buckets ($t_j \leftarrow t_j + 1$) vergrößert und mit $e_{i+1}$ fortgeführt.

- Fall 2: $i \notin [s_j, t_j)$

  $e_i$ ist im falschen Bucket. also wird $e_i$ mit $e_{t_j}$ getauscht, das Ende des Buckets $b_j$ vergrößert und mit $e_i$ fortgeführt.

Diese Iteration endet nach $2n$ Schritten. Hat ein Bucket mehr als $l$ Elemente, so wird auf diesem erneut der MSD-Radix aufgerufen. Entsprechend wird die nächste Stelle angeschaut, hier implementiert durch das Verschieben der Maske um $d * k$ Bits nach links, wobei $d$ die entsprechende Rekursionstiefe ist. Erneute Aufrufe von MSD-Radix werden durch einen Threadpool parallel abgearbeitet.

## Robin-Hood-Sort

Das Element $e_i$ wird

# Optimierungen

1. Multithreading:

   Anstelle immer einen neuen Thread zu erstellen, wird ein Threadpool verwendet, um so Aufwand zu sparen.

   Um den Aufwand der Parallelisierung bei nur einem Thread zu vermeiden, wird ein Sentinel-TaskHandler eingesetzt, der Aufgaben direkt erledigt.

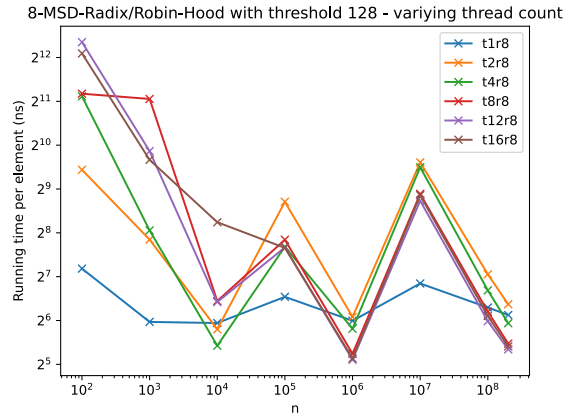
   Da ein MSD-Radix Durchlauf schnell erledigt ist, hat jeder Thread eine eigene Task-Queue um Synchronisierungsoverhead durch Mutexe zu reduzieren. Erst dadurch war die parallele Version schneller als die sequentielle.

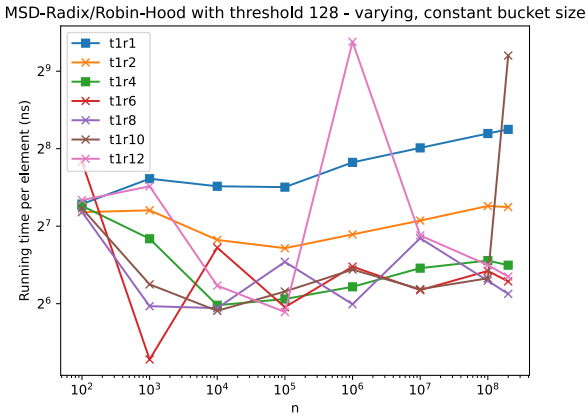
2. MSD-Radix:

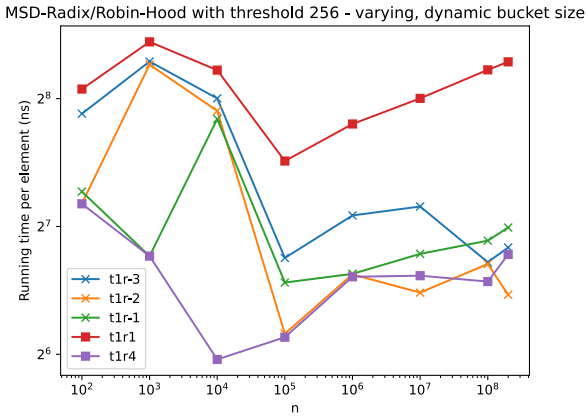
   Fall 1 wie oben erzwing, dass jedes Element bis zu zwei mal angeschaut werden muss. Stattdessen kann direkt zum Ende des zugehörigen Buckets gesprungen werden, da die Elemente per Induktion schon richtig einsortiert sind.

   Durch die Experimente sieht man, wie für statisches $k$ die Laufzeit bei verschiedenem $n$ variiert. Eine dynamische Wahl in Abhängigkeit der Anzahl an Elementen könnte ein besseres Resultat erbringen.

# Diagramme


8-MSD-Radix/Robin-Hood with threshold 128 - variying thread count
Running time per element (ns)
n
t1r8
t2r8
t4r8
t8r8
t12r8
t16r8


MSD-Radix/Robin-Hood with threshold 128 - varying, constant bucket size
Running time per element (ns)
n
t1r1
t1r2
t1r4
t1r6
t1r8
t1r10
t1r12


MSD-Radix/Robin-Hood with threshold 256 - varying, dynamic bucket size
Running time per element (ns)
n
t1r-3
t1r-2
t1r-1
t1r1
t1r4