Gruppe: 15 Luis Nothvogel, Tobias Schoch

17.2

Bei Verwendung der WORST fit Policy fällt auf, dass es ein Fragment mehr in der „free list“ gibt (Blau umrandet). Dies passiert durch die Verwendung des größten freien Fragmentes in der „free list“. In diesem Fall wird nicht ein anderes passendes Fragment ausgewählt, sondern immer das größte und das war bei diesem Beispiel die verbleibenden 76 bzw. 74 Byte. Dadurch entsteht ein weiteres Element in der „free list“.

```
ptr[3] = Alloc(8)  returned 1016 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1024 sz:76 ]

Free(ptr[3]) returned 0
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [ addr:1024 sz:76 ]

ptr[4] = Alloc(2)  returned 1024 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [ addr:1026 sz:74 ]

ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [ addr:1033 sz:67 ]
```

Worst fit

```
Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 3 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

Best fit

17.3

Gemäß der „FIRST fit Policy“ wird in der „free list“ immer das erste passende Fragment gewählt. Das kann man auch in der Ausgabe beobachten. Die „FIRST fit Policy“ ist wesentlicher schneller, was sich in der Anzahl der durchsuchten Elemente wiederspiegelt. Bei der WORST Policy und der BEST Policy wurde durch jedes Element gegangen, während bei der FIRST Fit Policy lediglich das erste passende verwendet wird. Deshalb haben wir auch bei WORST zum Schluss jeweils 4 durchsuchte Elemente und bei FIRST Fit weniger.

```
Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 3 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

First fit

```
ptr[3] = Alloc(8)  returned 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

Best fit

17.4

Bei der Listensortierung SIZESORT+ sowie ADDRSORT ist die Free Liste der Größe nach aufsteigend sortiert. Also das größte Fragment ist am Ende der Liste.

Bei der Listensortierung SIZESORT- ist die Free Liste der Größe nach absteigend sortiert. Also das kleinste Fragment ist am Ende der Liste.

Einfluss von SIZESORT+ und ADDRSORT:

- FIRST Fit: Keinerlei Einfluss

FIRST FIT

```
ptr[5] = Alloc(7)  returned 1008 (searched 3 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

Einfluss von SIZESORT-:

- FIRST Fit: Die Blöcke sind nun in der perfekten Reihenfolge, also mit dem Größten Element zuerst. Dadurch kann FIRST Fit logischerweise enorme Zeit einsparen.

FIRST Fit

```
ptr[5] = Alloc(7)  returned 1026 (searched 1 elements)
Free List [ Size 5 ]:  [ addr:1033 sz:67 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]
```

Die Listensortierungen haben keinerlei Einfluss auf BEST und WORST Fit, da diese selbst noch alle Blöcke durchgehen und diese nochmal für sich selbst sortieren. Daher ist die Geschwindigkeit und die Größe der Free Liste gleichgeblieben.

Best fit – keine Änderung bei der Größe der Free Liste oder der Geschwindigkeit

```
ptr[4] = Alloc(2)  returned 1000 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]
```

```
ptr[4] = Alloc(2)  returned 1000 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1016 sz:84 ] [ addr:1008 sz:8 ] [ addr:1003 sz:5 ] [ addr:1002 sz:1 ]
```

Worst fit – keine Änderung bei der Größe der Free Liste oder der Geschwindigkeit

```
ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1033 sz:67 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]
```

```
ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [ addr:1033 sz:67 ]
```

17.5

Nach 1000 Ausführungen kann es möglich sein, dass man sehr viele kleine Blöcke hat. Da diese stets aufgesplittet werden, aber nicht mehr zusammengefügt werden. Mit Coalescing ist dies jedoch möglich, da sich Nachbarblöcke verknüpfen können um die external Fragmentation zu vermindern. Durch die Verknüpfung ist es nun auch einfacher für größere Reservierungen einen freien Platz zu bekommen. Auch für die Schnelligkeit ist es von großem Vorteil, da die „Free List“ weniger Elemente hat, die nach passender Größe durchsucht werden müssen.

Unten in den Schaubildern kann man schön sehen, was für einen Unterschied in der „Free List“ diese Technik hat.

```
ptr[516] = Alloc(10)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1010 sz:90 ]
```

Mit Coalescing

```
Free(ptr[598]) returned 0
Free List [ Size 51 ]:  [ addr:1000 sz:1 ] [ addr:1001 sz:1 ] [ addr:1002 sz:1 ] [ addr:1003 sz:1 ] [ addr:1004 sz:1 ] [ addr:1005 sz:2 ] [ addr:1007 sz:1 ] [ addr:1008 sz:2 ] [ addr:1010 sz:1 ] [ addr:1011 sz:2 ] [ addr:1013 sz:1 ] [ addr:1014 sz:1 ] [ addr:1015 sz:1 ] [ addr:1016 sz:3 ] [ addr:1019 sz:1 ] [ addr:1020 sz:1 ] [ addr:1021 sz:1 ] [ addr:1022 sz:3 ] [ addr:1025 sz:3 ] [ addr:1028 sz:1 ] [ addr:1029 sz:2 ] [ addr:1031 sz:1 ] [ addr:1032 sz:2 ] [ addr:1034 sz:4 ] [ addr:1038 sz:1 ] [ addr:1039 sz:1 ] [ addr:1040 sz:1 ] [ addr:1041 sz:1 ] [ addr:1042 sz:2 ] [ addr:1044 sz:4 ] [ addr:1048 sz:1 ] [ addr:1049 sz:3 ] [ addr:1052 sz:1 ] [ addr:1053 sz:4 ] [ addr:1057 sz:1 ] [ addr:1058 sz:1 ] [ addr:1059 sz:2 ] [ addr:1061 sz:1 ] [ addr:1062 sz:4 ] [ addr:1066 sz:5 ] [ addr:1071 sz:1 ] [ addr:1072 sz:5 ] [ addr:1077 sz:1 ] [ addr:1078 sz:1 ] [ addr:1079 sz:5 ] [ addr:1084 sz:2 ] [ addr:1086 sz:1 ] [ addr:1087 sz:1 ] [ addr:1088 sz:6 ] [ addr:1094 sz:2 ] [ addr:1096 sz:4 ]
```

Ohne Coalescing

Wenn man die entsprechenden verschiedenen Policies ausführt, dann sieht man direkt was für eine große Effizienz Coalescing hat.

Ohne Coalescing

FIRST List Size: 51

```
./malloc.py -n 1000 -H 0 -p FIRST -s 0 -c
```

```
Free(ptr[598]) returned 0
Free List [ Size 51 ]:  [ addr:1000 sz:1 ] [ addr:1001 sz:1 ] [ addr:1002 sz:1 ] [ addr:1003 sz:1 ] [ addr:1004 sz:1 ] [ addr:1005 sz:2 ] [ addr:1007 sz:1 ] [ addr:1008 sz:2 ] [ addr:1010 sz:1 ] [ addr:1011 sz:2 ] [ addr:1013 sz:1 ] [ addr:1014 sz:1 ] [ addr:1015 sz:1 ] [ addr:1016 sz:3 ] [ addr:1019 sz:1 ] [ addr:1020 sz:1 ] [ addr:1021 sz:1 ] [ addr:1022 sz:3 ] [ addr:1025 sz:3 ] [ addr:1028 sz:1 ] [ addr:1029 sz:2 ] [ addr:1031 sz:1 ] [ addr:1032 sz:2 ] [ addr:1034 sz:4 ] [ addr:1038 sz:1 ] [ addr:1039 sz:1 ] [ addr:1040 sz:1 ] [ addr:1041 sz:1 ] [ addr:1042 sz:2 ] [ addr:1044 sz:4 ] [ addr:1048 sz:1 ] [ addr:1049 sz:3 ] [ addr:1052 sz:1 ] [ addr:1053 sz:4 ] [ addr:1057 sz:1 ] [ addr:1058 sz:1 ] [ addr:1059 sz:2 ] [ addr:1061 sz:1 ] [ addr:1062 sz:4 ] [ addr:1066 sz:5 ] [ addr:1071 sz:1 ] [ addr:1072 sz:5 ] [ addr:1077 sz:1 ] [ addr:1078 sz:1 ] [ addr:1079 sz:5 ] [ addr:1084 sz:2 ] [ addr:1086 sz:1 ] [ addr:1087 sz:1 ] [ addr:1088 sz:6 ] [ addr:1094 sz:2 ] [ addr:1096 sz:4 ]
```

BEST List Size: 31

```
./malloc.py -n 1000 -H 0 -p BEST -s 0 -c
```

```
Free List [ Size 31 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:1 ] [ addr:1006 sz:1 ] [ addr:1007 sz:1 ] [ addr:1008 sz:5 ] [ addr:1013 sz:1 ] [ addr:1014 sz:1 ] [ addr:1015 sz:1 ] [ addr:1016 sz:5 ] [ addr:1021 sz:1 ] [ addr:1022 sz:3 ] [ addr:1031 sz:1 ] [ addr:1032 sz:2 ] [ addr:1034 sz:3 ] [ addr:1037 sz:4 ] [ addr:1041 sz:1 ] [ addr:1042 sz:2 ] [ addr:1052 sz:1 ] [ addr:1053 sz:6 ] [ addr:1059 sz:2 ] [ addr:1061 sz:1 ] [ addr:1068 sz:1 ] [ addr:1069 sz:3 ] [ addr:1072 sz:5 ] [ addr:1077 sz:3 ] [ addr:1080 sz:1 ] [ addr:1081 sz:5 ] [ addr:1086 sz:3 ] [ addr:1089 sz:5 ] [ addr:1094 sz:2 ] [ addr:1096 sz:4 ]
```

WORST List Size: 100

```
./malloc.py -n 1000 -H 0 -p WORST -s 0 -c
```

```
ptr[828] = Alloc(9)  returned -1 (searched 100 elements)
Free List [ Size 100 ]:  [ addr:1000 sz:1 ] [ addr:1001 sz:1 ] [ addr:1002 sz:1 ] [ addr:1003 sz:1 ] [ addr:1004 sz:1 ] [ addr:1005 sz:1 ] [ addr:1006 sz:1 ] [ addr:1007 sz:1 ] [ addr:1008 sz:1 ] [ addr:1009 sz:1 ] [ addr:1010 sz:1 ] [ addr:1011 sz:1 ] [ addr:1012 sz:1 ] [ addr:1013 sz:1 ] [ addr:1014 sz:1 ] [ addr:1015 sz:1 ] [ addr:1016 sz:1 ] [ addr:1017 sz:1 ] [ addr:1018 sz:1 ] [ addr:1019 sz:1 ] [ addr:1020 sz:1 ] [ addr:1021 sz:1 ] [ addr:1022 sz:1 ] [ addr:1023 sz:1 ] [ addr:1024 sz:1 ] [ addr:1025 sz:1 ] [ addr:1026 sz:1 ] [ addr:1027 sz:1 ] [ addr:1028 sz:1 ] [ addr:1029 sz:1 ] [ addr:1030 sz:1 ] [ addr:1031 sz:1 ] [ addr:1032 sz:1 ] [ addr:1033 sz:1 ] [ addr:1034 sz:1 ] [ addr:1035 sz:1 ] [ addr:1036 sz:1 ] [ addr:1037 sz:1 ] [ addr:1038 sz:1 ] [ addr:1039 sz:1 ] [ addr:1040 sz:1 ] [ addr:1041 sz:1 ] [ addr:1042 sz:1 ] [ addr:1043 sz:1 ] [ addr:1044 sz:1 ] [ addr:1045 sz:1 ] [ addr:1046 sz:1 ] [ addr:1047 sz:1 ] [ addr:1048 sz:1 ] [ addr:1049 sz:1 ] [ addr:1050 sz:1 ] [ addr:1051 sz:1 ] [ addr:1052 sz:1 ] [ addr:1053 sz:1 ] [ addr:1054 sz:1 ] [ addr:1055 sz:1 ] [ addr:1056 sz:1 ] [ addr:1057 sz:1 ] [ addr:1058 sz:1 ] [ addr:1059 sz:1 ] [ addr:1060 sz:1 ] [ addr:1061 sz:1 ] [ addr:1062 sz:1 ] [ addr:1063 sz:1 ] [ addr:1064 sz:1 ] [ addr:1065 sz:1 ] [ addr:1066 sz:1 ] [ addr:1067 sz:1 ] [ addr:1068 sz:1 ] [ addr:1069 sz:1 ] [ addr:1070 sz:1 ] [ addr:1071 sz:1 ] [ addr:1072 sz:1 ] [ addr:1073 sz:1 ] [ addr:1074 sz:1 ] [ addr:1075 sz:1 ] [ addr:1076 sz:1 ] [ addr:1077 sz:1 ] [ addr:1078 sz:1 ] [ addr:1079 sz:1 ] [ addr:1080 sz:1 ] [ addr:1081 sz:1 ] [ addr:1082 sz:1 ] [ addr:1083 sz:1 ] [ addr:1084 sz:1 ] [ addr:1085 sz:1 ] [ addr:1086 sz:1 ] [ addr:1087 sz:1 ] [ addr:1088 sz:1 ] [ addr:1089 sz:1 ] [ addr:1090 sz:1 ] [ addr:1091 sz:1 ] [ addr:1092 sz:1 ] [ addr:1093 sz:1 ] [ addr:1094 sz:1 ] [ addr:1095 sz:1 ] [ addr:1096 sz:1 ] [ addr:1097 sz:1 ] [ addr:1098 sz:1 ] [ addr:1099 sz:1 ]
```

Mit Coalescing

FIRST List Size: 1

```
./malloc.py -n 1000 -H 0 -p FIRST -s 0 -C -c
```

```
ptr[516] = Alloc(10)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1010 sz:90 ]
```

BEST List Size: 1

```
./malloc.py -n 1000 -H 0 -p BEST -s 0 -C -c
```

```
ptr[514] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1002 sz:98 ]
```

WORST List Size: 1

```
./malloc.py -n 1000 -H 0 -p WORST -s 0 -C -c
```

```
Free(ptr[516]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]
```

Die Sortierung der Liste ist nur für die FIRST fit Policy relevant, sowie die Next Fit Policy, da diese lediglich das erste bzw. das nächste freie passende Element verwenden, während die Anderen zuerst jedes Mal für sich selber „sortieren“ was der größte bzw. kleinste Block ist.

17.6

-P als Flag beschreibt die Wahrscheinlichkeit, dass eine zufällige Aktion eine Speicherreservierung ist. Desto höher die Wahrscheinlichkeit, desto niedriger ist die Wahrscheinlichkeit, dass es kein free wird. Das heißt mit einem Wert von 75, besteht eine Wahrscheinlichkeit von 75%, dass eine Aktion eine Speicherreservierung ist.

Wenn die Wahrscheinlichkeit auf 1 gesetzt wird, dann führt das Programm zuerst eine Speicherreservierung aus, da es noch kein Free ausführen kann. Danach wird mit einer Wahrscheinlichkeit von 99% free ausgeführt. Danach findet eine Speicherreservierung statt, da wieder kein free ausgeführt werden kann usw. Man kann ein schönes Pattern erkennen.

```
ptr[0] = Alloc(5)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1005 sz:95 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:5 ] [ addr:1005 sz:95 ]

ptr[1] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 2 ]:  [ addr:1002 sz:3 ] [ addr:1005 sz:95 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:3 ] [ addr:1005 sz:95 ]

ptr[2] = Alloc(9)  returned 1005 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:3 ] [ addr:1014 sz:86 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:3 ] [ addr:1005 sz:9 ] [ addr:1014
sz:86 ]

ptr[3] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 3 ]:  [ addr:1002 sz:3 ] [ addr:1005 sz:9 ] [ addr:1014 sz:86 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:3 ] [ addr:1005 sz:9 ] [ addr:1014
sz:86 ]

ptr[4] = Alloc(4)  returned 1005 (searched 3 elements)
Free List [ Size 4 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:3 ] [ addr:1009 sz:5 ] [ addr:1014
sz:86 ]

Free(ptr[4]) returned 0
Free List [ Size 5 ]:  [ addr:1000 sz:2 ] [ addr:1002 sz:3 ] [ addr:1005 sz:4 ] [ addr:1009
sz:5 ] [ addr:1014 sz:86 ]
```

Wenn die Wahrscheinlichkeit auf 99 gesetzt wird, dann führt das Programm bei einer Aktion zu 99% eine Speicherreservierung aus und reserviert Speicher. Hier kann man schön sehen, wie 10-mal Speicher reserviert wird, aber kein einziges Mal Speicher befreit wird.

```
ptr[0] = Alloc(8)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1008 sz:92 ]

ptr[1] = Alloc(3)  returned 1008 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1011 sz:89 ]

ptr[2] = Alloc(5)  returned 1011 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1016 sz:84 ]

ptr[3] = Alloc(4)  returned 1016 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1020 sz:80 ]

ptr[4] = Alloc(6)  returned 1020 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1026 sz:74 ]

ptr[5] = Alloc(6)  returned 1026 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1032 sz:68 ]

ptr[6] = Alloc(8)  returned 1032 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1040 sz:60 ]

ptr[7] = Alloc(3)  returned 1040 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1043 sz:57 ]

ptr[8] = Alloc(10)  returned 1043 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1053 sz:47 ]

ptr[9] = Alloc(10)  returned 1053 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1063 sz:37 ]
```

17.7

./malloc.py +25,+25,+25,+25,-0,-1,-2,-3 -c

Mit dieser Anfrage für die Speicherreservierung haben wir nun die Größe der „Free List“ vervierfacht. In der Liste sind nun 4 Elemente mit jeweils einer Größe von 25, die im Anschluss direkt befreit werden mit free (-0,-1,-2,-3).

```
Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:25 ] [ addr:1025 sz:25 ] [ addr:1050 sz:25 ] [ addr:1075 sz:25 ]
```

Ich interpretiere die Aufgabe so, dass wir die abzuarbeitende Liste von dem obigen Beispiel verwenden sollen:

./malloc.py +25,+25,+25,+25,-0,-1,-2, -3 -c -C

```
Free(ptr[0]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:25 ]

Free(ptr[1]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:50 ]

Free(ptr[2]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:75 ]

Free(ptr[3]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]
```

Wenn wir nun mit Coalescing arbeiten (-C), dann haben wir stets nur ein Element in der „Free List“. Dank Coalescing verbinden sich stets die Nachbarelemente, sofern diese Frei sind.

./malloc.py +25,+25,+25,+25,-0,-1,-2, -3 -c -C -l SIZESORT+

./malloc.py +25,+25,+25,+25,-0,-1,-2, -3 -c -C -l SIZESORT-

Mit “SIZESORT+/-“ oder „ADDRSORT“ zu arbeiten erübrigt sich hier, da alle Elemente gleich groß sind. Daher erledigt sich auch BEST, WORST und FIRST, da alle stets das erste Element bekommen und dieses dann auch frei ist.