(a)

Algorithmus 1: ExtrempunktSort

```
Eingabe: Zahlenliste A
Ausgabe: Sortierte Permutation von A
X ← Indizes der Extrempunkte in aufsteigender Reihenfolge
if |X| = 2 and A[0] < A[X[1]] then
    return A
end
s ← true
for i ← 0; i < |X| − 2; i += 2 do
    if s then
        if A[X[i]] > A[X[i + 1]] then
            Reverse(A von X[i] bis X[i + 1]);
        else
            Reverse(A von X[i + 1] bis X[i + 2]);
        end
        Merge(A,X[i], X[i + 1], X[i + 2]);
    else
        if A[X[i]] < A[X[i + 1]] then
            Reverse(A von X[i] bis X[i + 1]);
        else
            Reverse(A von X[i + 1] bis X[i + 2]);
        end
        MergeReverse(A,X[i], X[i + 1], X[i + 2]);
    end
    s ← ¬s;
end
return ExtrempunktSort(A);
```

Annahme: Für $|X| > 2$ ist $|X|$ ungerade, damit durch die For-Schleife das gesamte Feld abgedeckt wird. Falls $|X|$ zu beginn gerade ist, kann ein beliebiges Element aus X dupliziert werden, damit der Algorithmus korrekt arbeitet.

Weiterhin ist mit Merge die gleichnamige Subroutine aus der Vorlesung gemeint. MergeReverse beschreibt dieselbe Funktion, mit dem Unterschied, dass dort zwei absteigend sortierte Listen zu einer absteigend sortierten Liste zusammengeführt werden.

(b)

Die Korrektheit des Algorithmus soll mithilfe einer Invariante gezeigt werden.

Invariante: Zu Beginn des Algorithmus besteht die Liste aus abwechselnd aufsteigend und absteigend sortierten Teillisten. Diese Teillisten werden jeweils von zwei benachbarten Extrempunkten abgegrenzt.

Bei jedem rekursiven Aufruf von ExtrempunktSort wird die Anzahl dieser Teillisten (aufgerundet) halbiert: Pro Durchlauf der For-Schleife werden zwei der sortierten Teillisten zu einer sortierten Teilliste zusammengeführt - je nachdem, ob s gerade *true* ist, ist die resultierende Teilliste entweder aufsteigend oder absteigend.

Soll die resultierende Liste aufsteigend sortiert sein, wird zunächst dafür gesorgt, dass beide behandelten Teillisten ebenfalls aufsteigend sortiert sind, indem die absteigend sortierte Liste umgekehrt wird. Diese beiden Teillisten können dann mit Merge zusammengeführt werden. Soll absteigend sortiert werden, passiert genau der entgegengesetzte Prozess.

Da am Ende der For-Schleife s umgekehrt wird, werden beim nächsten Durchlauf die nächsten beiden Teillisten entgegengesetzt sortiert, sodass die Invariante erhalten bleibt.

Da laut der Annahme aus (a) $|X|$ ungerade ist, ist die Gesamtanzahl der Teillisten gerade. Da durch die For-Schleife immer genau zwei Teillisten zusammengefügt werden, ist garantiert, dass das gesamte Feld abgearbeitet wird und somit die Invariante für die komplette Liste gilt.

Die erste Teilliste wird immer aufsteigend sortiert. Sobald also am Ende eines Durchlaufs die komplette Liste nur noch aus einer Teilliste besteht (d.h. die Anzahl der Extrempunkte ist 2), ist die vollständige Liste sortiert und der Algorithmus terminiert nach dem unmittelbar folgenden Aufruf.

(c)

Ein einzelner rekursiver Aufruf von ExtrempunkSort hat die Laufzeit $\Theta(n)$:

- Die Bestimmung der Extrempunkte X liegt in $\Theta(n)$, da jedes $a \in A$ genau einmal mit seinen jeweiligen Nachbarn verglichen werden muss.
- Innerhalb eines Durchlaufs der For-Schleife werden mit $k = |M|$ genau zwei Teilprobleme der Größe $\frac{n}{k}$ in Linearzeit gelöst (Annahme: Reverse kehrt eine Liste der Größe n in $\Theta(n)$ um, indem die ersten $\lfloor n/2 \rfloor$

Elemente genau einmal mit ihrem Gegenüber getauscht werden). MERGE bzw. MERGEREVERSE benötigen ebenfalls lineare Zeit (siehe Vorlesung). Die Schleife wird insgesamt $k/2$-mal ausgeführt, somit beträgt die Gesamtzeit $\Theta(\frac{k}{2} \cdot \frac{2n}{k}) = \Theta(n)$.

- Gesamtzeit eines Durchlaufs: $\Theta(n) + \Theta(n) = \Theta(n)$

Die Anzahl der rekursiven Aufrufe hängt lediglich von der Anzahl der Extrempunkte k ab: der Algorithmus terminiert erst dann, wenn $k = 2$. Wie in Teilaufgabe (b) gezeigt, halbiert sich die Anzahl der sortierten Teillisten (und damit auch k) mit jedem Aufruf.

Somit ist die Anzahl der Aufrufe $\log_2 k = \Theta(\log k)$. Dies multipliziert mit der Laufzeit pro Durchlauf ergibt $\Theta(n \log k)$.