שיטת Andrew (Monotone Chain)

שיטת Andrew היא אלגוריתם יעיל לחישוב מעטפת קמורה עבור קבוצת נקודות.
היא פועלת כך:

1. **ממיינת את הנקודות לפי x ואחר כך y**
2. **בונה את החצי התחתון** של המעטפת תוך שימוש בהיגיון של מחסנית
3. **בונה את החצי העליון** באותו אופן
4. מאחדת את שתי הרשימות

המבנה הפנימי שמשמש אותנו למיון הנקודות ובניית המחסנית ה – hull-הוא מבנה הנתונים שמעניין אותנו לניתוח יעילות.

```
===== GPROF Efficiency Report =====
[list]
Average: 1.978 seconds

[deque]
Average: 1.44 seconds

[vector]
Average: 0.758 seconds

>> Based on final cumulative time in gprof output (total run time approximation)
```

השוואת ביצועים vector vs deque vs list :

תכונה	std::list	std::deque	std::vector
מיון	❌ איטי מאוד (צריך להמיר או להשתמש ב־ list::sort)	✅ מהיר יחסית	✅ מהיר מאוד
גישה לפי אינדקס	❌ O(n)	✅ O(1) לרוב	✅ O(1)
הוספה/הסרה בקצה	✅ מהירה	✅ מהירה	✅ מהירה
איטרציה ליניארית	❌ איטית מאוד (גישה pointer-based)	✅ טובה	✅ מהירה
שימוש בזיכרון	❌ מאוד לא יעיל	❌ מפוזר	✅ רציף

ניתוח ביצועים בשיטת Andrew

1. **std::vector הכי יעיל לרוב**
   - תומך ב־ std::sort ישירות (יעיל מאוד)
   - מוסיף/מסיר בקצה ב־ O(1) אמיתי
   - רציף בזיכרון - > cache-friendly
   - מתאים למחסנית
2. **std::deque יעיל אך מעט פחות**
   - תומך ב־std::sort (אבל איטי יותר כי לא רציף בזיכרון)
   - מתאים למחסנית (push_back, pop_back)
   - ביצועים קרובים ל־ vector אך מעט נחותים
3. **std::list הכי פחות יעיל**
   - לא ניתן למיין עם std::sort אלא עם list::sort (אלגוריתם פנימי איטי יותר)
   - גישה איטית מאוד לכל איבר (אין אינדקס)
   - צריכת זיכרון גבוהה מאוד (pointer לכל איבר)
   - מתאים אם חייבים הסרה/הוספה מכל מקום – לא המצב כאן

✅ **מסקנה:**

- **std::vector** הוא הבחירה המומלצת לחישוב מעטפת קמורה בשיטת Andrew
  - מיון מהיר
  - גישה מהירה
  - שימוש בזיכרון רציף
- **std::deque** קרוב ל־ vector אך נחות בזכות מימוש פחות cache-friendly
- **std::list** מתאים רק למקרים בהם צריך להסיר מהאמצע – לא רלוונטי כאן ,ולכן הכי איטי